

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES**

*Reconfigurable Computing Based on Commercial
FPGAs. Solutions for the Design and
Implementation of Partially Reconfigurable
Systems*

*Computación reconfigurable basada en FPGAs
comerciales. Soluciones para el diseño e
implementación de sistemas parcialmente
reconfigurables.*

TESIS DOCTORAL

Yana Esteves Krasteva

Máster en Ingeniería Electrónica, Universidad Técnica de Sofía, Bulgaria

2009

*DEPARTAMENTO DE AUTOMÁTICA, INGENIERÍA ELECTRÓNICA E
INFORMÁTICA INDUSTRIAL
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES*

Reconfigurable Computing Based on Commercial FPGAs.
Solutions for the Design and Implementation of Partially
Reconfigurable Systems

Computación reconfigurable basada en FPGAs comerciales.
Soluciones para el diseño e implementación de sistemas
parcialmente reconfigurables.

TESIS DOCTORAL

Autor: Yana Esteves Krasteva

Máster en Ingeniería Electrónica, Universidad Técnica de Sofía.

Director: Eduardo de la Torre Arnanz

Doctor Ingeniero Industrial por la Universidad Politécnica de Madrid.

2009

Tribunal

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día de julio de 2009.

Presidente: Dr. Javier Uceda Antolín, Universidad Politécnica de Madrid.

Vocal: Dr. Manfred Glesner, Technische Universität Darmstadt.

Vocal: Dr. Juan Carlos López López, Universidad de Castilla-La Mancha.

Vocal: Dr. João Canas Ferreira, Universidade do Porto.

Secretario: Dr. Félix B. Tobajas Guerrero, Universidad de Las Palmas de Gran Canaria.

Suplente: Dr. Celia López Ongil, Universidad Carlos III de Madrid.

Suplente: Dr. Ángel de Castro Martín, Universidad Autónoma de Madrid.

Realizado el acto de defensa y lectura de la tesis el día 27 de julio de 2009 en la E.T.S. Ingenieros Industriales.

CALIFICACIÓN:

EL PRESIDENTE

LOS VOCALES

EL SECRETARIO

A Vladimir

A mi Mami

Agradecimientos

Llegados al punto de rellenar esta página, he querido hacer constancia de mis pensamientos y reconocimientos. Si tengo que resumir todas las experiencias vividas en una frase esta sería: “Realmente es duro pero, a pesar de todo, vale la pena”. Para mí, la valoración de todos estos años es positiva, aunque podría escribir otra tesis proponiendo mejoras (y eso que lo de escribir no es mi fuerte...).

A Vladimir: За това че измина този дълъг път с мен. За подкрепата и доверието. *Благодаря ти!*

A mi Abuela: За безценните учения, и затова че винаги бе, си и ще бъдеш неотлъчно до мен. Благодаря ти бабо, жалко че не ни дочака. *Благодаря ти!*

A mi Madre: Por trasmitirme su fuerza, darme no sólo oportunidades en la vida, sino también enseñarme a aprovecharlas. Благодаря ти майко от все сърце. Mis agradecimientos a Javi Checa por su infinita bondad y cariño y por ser un contrapeso. *Благодаря ви!*

A mi Hermana Irina: Por ser un apoyo y un ejemplo importante durante mi estancia en España. A su mari-novio Javi Gordo por los agradables momentos. *Благодаря ви!*

A mi Hermana Tania: Por ser un apoyo en la distancia y aguantar muchas tormentas en nombre de todos, always with Joost to whom I am also thankful. *Благодаря ви!*

A mi Padre, espero tu reencuentro.

Le agradezco mi Tutor de tesis Eduardo de la Torre, por su paciencia durante el largo proceso de corrección del documento y por su bondad. *Gracias!*

Les agradezco a todos los Profesores del departamento, muy especialmente a Teresa Riesgo, por ser la "culpable" de que haya venido a España y siempre haber apostado por mí y a Jorge por ser mi compi. *Gracias!*

Les agradezco a mis Compañeros, especialmente a Carmen por ayudarme con el “Inglés” y por su apoyo. A Víctor, Leo, Andrés, Rubén, David, Dani y Miroslav. Las risas, las cervezas del Chiri y las reuniones en casas, como fuente de ánimos, fuerza y superación, han sido algo fundamental durante la tesis. También a Ángel, Felipe, Jaime, Ana B., Vincenzo, Benoit, Zoran y Cony. *Gracias!*

Thanks to the thesis reviewers, Leandro Soares Indrusiak from the University of York (England), Dirk Stroobandt from the University of Gent (Belgium) and José Silva Matos from the University of Porto (Portugal). *Thank you!*

También me gustaría agradecer a todas las personas dentro y fuera de la UPM y del CEI, especialmente a Yolanda y Nieves, que dan apoyo a la investigación. Durante los años de mi estancia en España he vivido algunas mejoras, tanto en la situación de los "precarios" como en las oportunidades y la calidad de la investigación, lo cual me gratifica. *Gracias!*

Last, but not least, I would like to thank to the open research community. I consider that to share openly the knowledge is the base of a quality research. In this context, special thanks to Departamento de Fundamentos da Computação, Pontifícia Universidade Católica do Rio Grande do Sul (Brasil), to the research group lead by Fremando Moraes and Ney Calazans. *Thank you!*

Resumen

Esta tesis doctoral está enmarcada en el campo de investigación de la computación reconfigurable. Este campo ha experimentado un crecimiento abrumador en los últimos años como resultado de la evolución de los dispositivos reconfigurables, donde las *Field Programmable Gate Arrays (FPGAs)* son el máximo exponente desde el punto de vista comercial.

De forma tradicional las empresas de electrónica han seleccionado las FPGAs como prototipos iniciales de productos de altas prestaciones. Luego el sistema final es integrado en *Application Specific Integrated Circuits (ASICs)* que se producen en grandes volúmenes perimiendo amortiza su alto coste de diseño y producción y aprovechando la ventaja del bajo coste por unidad. Por otro lado, los DSPs (Digital Signal Processing) y los microprocesadores han sido preferidos por su bajo coste ante las FPGAs el campo de los dispositivos con menores requisitos de cómputo.

En los últimos años, este panorama está sufriendo una serie de cambios. Ahora el mercado busca mas soluciones “reconfigurables” ya que permiten reducir el tiempo de salida del producto al mercado (*time-to-market*), aumentar el tiempo del producto en el mercado (*time-in-market*) y además cubren los amplios requisitos de cómputo. El cambio que se observa, se debe a que los dispositivos programables han evolucionado de simples estructuras programables a complejas plataformas reconfigurables. Las FPGAs del estado de la técnica han alcanzado un grado de integración muy alto y además ahora contienen, dentro de su arquitectura programable, microprocesadores y lógica específica de procesamiento digital de señal. Otro factor sumamente importante para el cambio es que las FPGAs permiten el diseño de dispositivos cuyo hardware pueden ser adaptado, o actualizado, una vez que el producto ya esta entregado e instalado, obteniendo así una flexibilidad en el hardware comparable con la del software, donde la actualización post-venta de los sistemas es una práctica muy explotada de cara a la reducción de costes y la salida rápida al mercado.

Por otro lado, y sobre todo en el ámbito académico, existen dispositivos reconfigurables con distinta granularidad que permites alcanzar altas prestaciones en comparación con las FPGAs comerciales de grano fino (comparable con la de los ASICs), pero están restringidas a una aplicación o grupo de aplicaciones. A pesar de que los dispositivos reconfigurables propietarios ofrecen muchas ventajas, esta opción ha sido descartada en la presente tesis debido a que, desde el punto de vista industrial requieren, aparte del diseño del ASIC reconfigurable, el desarrollo de un entorno de diseño completo. Todo esto conlleva a un elevado coste de recursos, además del alejamiento de las propuestas de la industria. **La presente tesis se ha centrado en proporcionar soluciones para dispositivos comerciales, FPGAs de grano fino**, con la finalidad de aprovechar las herramientas existentes y mantener las soluciones propuestas lo más cerca posible de la industria.

Los dispositivos reconfigurables proporcionan diversos métodos de reconfiguración, siendo el más atractivo la **reconfiguración parcial y dinámica**, ya que permite adaptar el dispositivo sin interrumpir su funcionamiento y crear dispositivos auto-adaptables. Este tipo de reconfiguración será el objeto de estudio de la tesis doctoral. La reconfiguración parcial permite tener una serie de tareas hardware (módulos que se ubican en la estructura reconfigurable) ejecutándose paralelamente en la FPGA y sustituir un bloque por otro, dependiendo de las necesidades del sistema, sin alterar el funcionamiento del resto de bloques. Esta idea básica en teoría brinda la flexibilidad del software al hardware, que combinado con su paralelismo implícito hace del sistema reconfigurable una potente herramienta que puede dar pie a la creación de sistemas adaptables o incluso auto-adaptativos, supercomputadores reconfigurables y hardware bio-inspirado entre otros. Por otro lado, a pesar que algunos proveedores de FPGAs permiten la reconfiguración parcial, el uso de esta técnica aún está restringido al ámbito académico y a sistemas muy básicos.

El trabajo de investigación descrito dentro de la presente tesis doctoral ha tenido por objeto el estudio de diversos aspectos de los sistemas parcialmente reconfigurables, la identificación de las principales deficiencias de las soluciones existentes y la propuesta de nuevas soluciones originales. Como resultado del estudio del estado del arte se ha visto que las soluciones existentes son poco flexibles y la escalabilidad de los sistemas que se pueden diseñar es reducida. **Por ello las propuestas originales de esta tesis tienen como objetivo permitir el diseño e implementación de sistemas parcialmente reconfigurables con alta escalabilidad y flexibilidad.**

La tesis principal del trabajo de investigación ha sido basada en la idea que para obtener una mayor flexibilidad de los sistemas se debe desligar el diseño del sistema reconfigurable del diseño de los cores que serán consumidos por dicho sistema.

La tesis doctoral ha contribuido proponiendo mejores soluciones a nivel de arquitectura, flujos de diseño y herramientas que han permitido el diseño e implementación de diversos sistemas parcialmente reconfigurables con distinto grado de flexibilidad y escalabilidad. La flexibilidad y la escalabilidad son términos que en los sistemas reconfigurables se pueden asociar a diversos aspectos. Dentro de esta tesis la flexibilidad está asociada principalmente a la diversidad de cores o tareas hardware que pueden ser consumidos o integrados en un sistema ya definido, mientras que la escalabilidad está referida al número de cores que pueden coexistir en el sistema y ser reconfigurados independientemente. Para poder diseñar sistemas flexibles y escalables, estas características deben estar cubiertas en distintos niveles. Más en detalle dentro de la presente tesis, desde el punto de vista de la arquitectura, la flexibilidad está cubierta por la posibilidad de posicionar libremente cores en una arquitectura escalable predefinida. Desde el punto de vista del sistema, la flexibilidad está reflejada por la posibilidad de no sólo de modificar o reconfigurar un core del sistema hardware, sino también de modificar las comunicaciones internas del mismo. Desde el punto de vista del dispositivo, la flexibilidad está garantizada por la transparencia en el proceso de reconfiguración. Por último, la flexibilidad en el proceso de diseño está cubierta por la definición de herramientas y flujos de diseño que permiten por un lado desligar el diseño del sistema reconfigurable del diseño de los cores para el sistema, y por otro lado que diseñadores sin conocimientos detallados de reconfiguración parcial puedan

diseñar cores. Dentro de la tesis doctoral se presentan cuatro dispositivos reconfigurable integrados en distintos entornos y con distinto grado de flexibilidad que corresponde al grado de aprovechamiento de las aportaciones originales de la tesis.

Las principales aportaciones de la tesis doctoral, relacionadas a cada uno de los aspectos mencionados en el párrafo anterior, y tratados en distintas partes de la tesis se resumen a continuación destacando en la medida de lo posible las diferencias con respecto al estado del arte:

- Se ha definido **una metodología de diseño de Arquitecturas Virtuales** (abstracción de la arquitectura física de la FPGA que incluye la distribución de los recursos programables en *slots* y la forma de interconexión de los slots). La metodología, propuesta originalmente en esta tesis, permite el diseño de sistemas reconfigurables con alta flexibilidad y escalabilidad comparadas con el estado del arte.
- **Una solución a la adaptación de las comunicaciones internas en los sistemas reconfigurables llamada DRNoC (Dynamic Reconfigurable NoC)**. La solución original abarca diversos aspectos e incluye la definición de una arquitectura de interconexión para los sistemas reconfigurables basada en redes en chip (Network on Chip - NoC), la definición de métodos de reconfiguración y el direccionamiento interno del sistema, y de forma más específica para las comunicaciones basadas en redes, la definición de un formato de tramas y la arquitectura de los enrutadores. La principal diferencia de la solución propuesta con el estado del arte es que DRNoC no restringe la comunicación únicamente a NoCs y permite la definición de cualquier tipo de esquema de comunicación (NoC, punto a punto, punto a multipunto, bus, o una combinación de las anteriores) y además, permite que varios esquemas de comunicación coexistan en el mismo sistema y que funcionen de forma independiente. De esta forma la solución propuesta brinda una mayor flexibilidad que las ya existentes.
- Se ha propuesto una **solución para la manipulación de los ficheros de configuración para las FPGA del tipo Virtex II/Pro que es la más completa** comparada con el estado del arte. Asimismo, una serie de herramientas que permiten la generación y extracción de cores para sistemas reconfigurables basados en FPGA Virtex II que ha sido la primera solución existente para estas FPGA.
- **Un flujo de diseño para cores basado en plantillas** que permite el diseño de cores hardware sin ser un experto en reconfiguración parcial y sin conocer los detalles del sistema final en el que se implementará el core.
- **El diseño, implementación y prueba de un sistema parcialmente reconfigurable basado en FPGAs comerciales de grano fino para redes de sensores**. La primera aproximación existente en el estado del arte al uso de los sistemas parcialmente reconfigurables en las redes de sensores.
- **La integración de un sistema reconfigurable en un entorno cliente-servidor** que incluye un original sistema de control de la reconfiguración.
- **Una solución para la depuración de los sistemas reconfigurables**.

- **Un sistema de emulación y prototipado rápido de las comunicaciones dentro de un chip** basado originalmente en la idea de la reutilización de cores hardware por medio de la técnica de reconfiguración parcial.

Como conclusión global del trabajo de investigación realizado cabe destacar que la presente tesis ha dado lugar a la creación y consolidación de una línea de investigación en el grupo de electrónica digital del Centro de Electrónica Industrial que actualmente se encuentra entre las más activas y de mayor importancia. Además, el trabajo de investigación y la divulgación de las aportaciones originales han permitido que el centro de investigación pase a formar parte del estado del arte de los sistemas parcialmente reconfigurables.

Abstract and Thesis Organization

The thesis is enclosed in the research area of reconfigurable computing which, in the last years, has experienced a remarkable growth as a result of the impressive evolution of reconfigurable devices. In this area, Field Programmable Gate Arrays (FPGAs) are the most outstanding representative from the commercial point of view.

Traditionally FPGAs have been used for prototyping, in previous to the final Application Specific Integrated Circuit (ASIC) design stages. However, the interest in the integration of FPGAs in final products has been growing in the last years. FPGAs are preferred for small production volumes, where the ASIC masks high cost is unaffordable and also in products where time-to-market is a priority, and waiting for a complete ASIC design cycle is not desirable.

State of the art FPGAs are highly integrated electronic circuits, composed of tens of millions of system gates, with competitive speed, performance and configurability. These devices have evolved from simple gate arrays to complex platforms that include embedded memory, multipliers and even microprocessors and digital signal processing elements. Additionally, the fine grain nature of the reconfigurable arrays, make FPGAs suitable for a broad set of application domains. On the other side, and mostly in the academic community, there are custom reconfigurable devices with different granularity levels that permit to achieve higher performance, compared to commercial FPGAs, but for a certain application domain. Although there are very good solutions in the academic state of the art, their main drawback from the industry point of view is that they require specific design environments and also, that the efforts and resources needed for designing such solutions are very high.

This thesis work is focused on providing solutions that target commercial fine grain reconfigurable devices, FPGAs, in order to take advantage of existing tools and to keep the proposed solutions closer to the industry.

Today FPGAs provide different reconfiguration options. Among them, the most challenging one is partial reconfiguration. This feature has special interest, as it permits system updates on the fly once the device is deployed, without the need of stopping it and without theoretical loss of performance. Partial reconfiguration is also an attractive feature because it permits to allocate different tasks/cores running in parallel in the device and change them on the fly as needed without disturbing other tasks/cores. This basic idea, brings software-like flexibility to hardware which, in combination with its inherited parallelism, opens the door for a broad amount of possibilities and applications, like runtime adaptive super-computing, adaptive embedded software

accelerators, bio-inspired, self-reconfigurable and self-arrangeable systems. However, even though some commercial FPGAs provide partial reconfiguration features, its utilization is still in its early stages and it is not well supported by FPGA vendors, making its exploitation in real electronic systems very difficult. Therefore, there are several academic groups working to provide alternative solutions for the design and implementation of partially reconfigurable systems based on commercial FPGAs that intend to stimulate their integration and use in the industry.

This research work intends to study different aspects of partially reconfigurable system on-chip and contribute with flexibility improvements. The main idea that will be followed along the thesis is that the design of reconfigurable systems will be considered an independent process from the design of cores that will be consumed by the system. This approach involves the design of flexible and scalable partial runtime reconfigurable systems, where most of the thesis contribution will be focused. More in detail, this thesis will contribute to improve architecture solutions, design tools and design flows of partially reconfigurable systems for commercial FPGAs and provide systems with higher flexibility and scalability.

Flexibility and scalability in a reconfigurable system are terms that can be related to several aspects. In this thesis flexibility is mainly related to the diversity of tasks or cores a system can consume, while scalability is connected to the number of cores that can run in parallel and be independently reconfigured. Flexibility and scalability have to be covered by the system at different levels and the work presented in this thesis will contribute in all the specific levels. More in detail, from an architectural point of view, flexibility is reflected by the possibility of freely loading tasks or cores in a defined, scalable architecture. From the system point of view, flexibility is related to the possibility of modifying not only the system functionality by loading different tasks, but also to adapt the on-chip communications. From the device point of view, flexibility is reflected by the reconfiguration process transparency and, from the design point of view, it is oriented to the definition of design tools and flows that will permit, as far as possible, non specialized designers to design cores for a partially reconfigurable system and without knowing the system details.

All the original proposed solutions, in each individual aspect, will be compared with the state of the art and complete systems solutions will be designed and will be integrated in different application domains in order to validate the thesis proposals.

In order to achieve better understanding of the thesis and to facilitate the comparison with some, selected, related work, the *thesis structure is not traditional. Instead of including a state of the art and a result Chapter, each Chapter is focused on a specific aspect of partially reconfigurable system design and includes state of the art and result sections.*

The first chapter, Chapter 1, introduces the main concepts to be used in the thesis. Chapter 2 is focused on reconfigurable systems architectures, contributing with architecture solutions and a design method. Chapter 3 proposes a solution that enhances the features of the architectures defined in Chapter 2, and provides more flexibility to the entire system by extending reconfiguration to the on-chip communication. Chapter 4 is related to the design flows and tools, where contributions are made in both aspects and the proposed solutions are compared with the state of the

art. Complete systems, with different independency levels, are presented in Chapter 5 in order to validate the thesis contributions. Conclusions, a summary of contributions and the future work are included in Chapter 6.

A more detailed description of each Chapter content is presented below:

Chapter 1 provides an introduction to the reconfigurable systems based on FPGAs topic, by first defining the place of FPGAs in the electronic industry and afterwards, introducing the main concepts to be used along the thesis. Although the focus is put on commercial reconfigurable devices, some custom reconfigurable systems are also described in order to have a complete view of the options in reconfigurable devices. The Chapter discusses the thesis main topic, related to partial runtime reconfigurable systems, highlighting its main advantages and disadvantages and, introducing some of the approaches to be followed in this thesis. The main term *introduced in this Chapter, associated to reconfigurable systems architectures, is "Virtual Architecture". The term defines the architecture of the partially reconfigurable system and how the different regions it is composed of are interconnected.* A brief summary of the thesis main goals is included at the end of the Chapter.

The main topic of Chapter 2 is related to reconfigurable systems architectures design. The Chapter includes a specific state of the art section that reviews some existing architecture solutions. After that, *a general method for virtual architectures design, an original thesis contribution, is presented in detail.* Afterwards, the method is applied to the design of general one dimensional (1D) and two dimensional (2D) architectures for Xilinx Virtex II/Pro FPGAs and, as an example, following the specific steps of the method, two 1D, bus based, architectures are designed. The architecture buses are compared with two state of the art solutions in terms of area and performance in the results section of the same Chapter.

Chapter 3 is focused on reconfigurable systems on-chip communication issues, where the need of adaptability is the main topic. Again, a state of the art description of some 2D reconfigurable systems is presented at the beginning of the Chapter and, afterwards, *an original solution, called Dynamic Reconfigurable NoC (DRNoC), is proposed.* This solution covers different aspects. First, an architecture oriented to support adaptability in the on-chip communications is originally proposed. The architecture is mapped to a Virtex II FPGA by modifying a virtual architecture from the general ones presented in Chapter 2. Second, two types of reconfigurations that span through different levels of the OSI communication model are originally proposed. Third, a set of Network on Chip models, focused on the communication adaptability are designed and/or adapted and presented in the Chapter, along with an original NoC packet format and router architecture. These models are mapped to the DRNoC architecture and implementation cost parameters are defined and used to evaluate different implementation options. Regarding the architecture reconfigurability, it is important to remark that along the entire Chapter, intermediate test of possible partial reconfigurations and test results are included. At the end of the Chapter, the proposed architecture is compared with the state of the art using a set of structural parameters taken from a reference work and complemented with others defined in the Chapter.

Chapter 4, focuses on the design tools and flows for partially reconfigurable systems. Again, an overview of the state of the art is included at the beginning of the Chapter.

Afterwards, *an original software solution for Virtex II configuration files (bitstreams) manipulations is presented.* The first part of the solution is a study of the Virtex II/Pro FPGA bitstream format, used to define a set of equations for accessing a specific bitstream resource (at register or block level). Based on these equations, a set of tools for bitstream manipulation that target resource restricted devices are originally presented. Also, a design flow, based on systems and virtual architectures templates, which permits a straightforward core design by non partial reconfiguration experts and without knowing the system details, is originally proposed.

In Chapter 5 four reconfigurable systems with different flexibility level, which corresponds to the level of the thesis proposals exploitation, are presented. The selected application domains attempt to demonstrate different advantages of the use of partial runtime reconfigurable systems and therefore are mainly a proof of concept. The first domain belongs to the *wireless sensor networks, where the possibility of sending new device configurations to a remote node is evaluated.* As far we are concern this is the first time a reconfigurable system is used in this application domain. The second, more sophisticated application is *a reconfigurable device integration, in an original way, in a client-server environment,* where new device updates are transparently loaded. The third one is an extension of the client-server system in order to support the insertion of debug modules through partial reconfiguration and, the last one is an, *originally proposed, on-chip communication emulation framework, where cores reusability is exploited through partial reconfiguration.*

Chapter 6 includes the thesis general conclusions and future work. The Chapter, also, answers to some main questions that appeared along the thesis and tries to evaluate the thesis contributions, which are summarized in the same Chapter.

Contents

Abstract and Thesis Organization	i
Table of Contents	x
List of Figures	xvii
List of Tables	xxi
1 Introduction	1
1.1 Technology Framework	3
1.2 Reconfiguration Techniques and Systems	5
1.3 Reconfigurable Devices	9
1.3.1 Commercial Reconfigurable Devices	9
1.3.1.1 Xilinx	9
1.3.1.2 Altera	13
1.3.1.3 Atmel	13
1.3.1.4 Lattice	14
1.3.2 Custom Reconfigurable Devices and Platforms	14
1.4 Partial Runtime Reconfiguration	16
1.5 Partially Reconfiguration Systems Based on Commercial FPGAs: Prospective	17
1.5.1 Dynamic Reconfigurable Commercial Devices: Selection	17
1.5.2 Hardware Abstraction	18
1.5.3 On-chip Communications	19
1.5.4 Design Methods and Software Support	20

1.6	Summary and thesis Objectives	22
2	Reconfigurable Systems for Commercial FPGAs	25
2.1	Introduction	27
2.2	Commercial Approach	28
2.2.1	FPGA Resource Division	28
2.2.2	Communication Structures	29
2.3	Academic Approaches	31
2.3.1	Lockwood-Horta et al.	31
2.3.2	Moraes-Calazans et al.	32
2.3.3	Platzner-Walder et al.	33
2.3.4	Nollet-Mignollet et al.	35
2.3.5	Hübner-Becker et al.	35
2.3.6	Erlangen Slot Machine (ESM)	37
2.4	Virtual Architectures for Partial RunTime Reconfigurable Systems. Design Method	39
2.4.1	Method Goal and General Steps	39
2.4.2	Method Guidelines for Xilinx FPGAs	42
2.4.2.1	System Reconfigurability Requirements and Target FPGAs	42
2.4.2.2	Logic Distribution Analysis	42
2.4.2.3	Resource Division	42
2.4.2.4	Slot Definition and in 1D and/or 2D Models	44
2.4.2.5	Communication Structure: Definition and Building	50
2.4.2.6	Point to Point Interconnections (P2P)	51
2.4.2.7	Point to Multi Point Interconnection (P2M)	53
2.4.3	Virtual Architectures Design Method Result. Definition files	54
2.4.4	Practical Application to 1D Partial Runtime Reconfigurable Systems	56
2.4.4.1	Bus Based Reconfigurable System for an XC2V1000 FPGA (Bus-v1)	56
2.4.4.2	Bus Based Reconfigurable System for an XC2V3000 FPGA (Bus-v2)	58
2.5	Results	59
2.6	Conclusions	62
3	Reconfigurable Networks on Chip for Reconfigurable Systems	63
3.1	Introduction	65

3.2	Academic Related Work	68
3.2.1	Nollet-Mignollet et al.	68
3.2.2	Hübner-Becker et al.	69
3.2.3	Bobda et al.-DyNoC	71
3.2.4	Pionteck et al. - CoNoChi	73
3.2.5	Moraes-Calazans et al.	74
3.3	Problem Statement	76
3.4	Selected Approach and Goals	76
3.5	Dynamic Reconfigurable NoC - DRNoC	77
3.5.1	Dynamic Reconfigurable NoC-Definition	77
3.5.2	DRNoC Mapping to a Xilinx Virtex II Virtual Architecture	79
3.5.3	DRNoC Design Resources	83
3.5.3.1	DRNoC Addressing and Packet Format	83
3.5.3.2	DRNoC Router Model	86
3.5.3.3	Traffic Generator Model	88
3.5.3.4	Traffic Receivers Models	88
3.5.3.5	Network Interface Models	89
3.5.3.6	Other Resources	89
3.5.4	DRNoC NoC Models Generation Tool: DRNoCGEN	89
3.5.5	DRNoC Reconfigurability	91
3.6	Reconfigurable NoCs Systems Evaluation Parameters	91
3.6.1	Performance Parameters	92
3.6.2	Structural Parameters	93
3.6.3	Cost Parameters	94
3.6.4	DRNoCs Design Resources Area Requirements	96
3.7	Reconfigurable NoC solutions for Partial Runtime Reconfigurable Systems Comparison	99
3.7.1	Structural Comparison	99
3.7.2	NoC Routers Comparison	100
3.7.3	NoC based Runtime Reconfigurable Systems General Comparison	103
3.8	General Evaluation of the Xmesh Communication Infrastructure	104
3.9	Conclusions	105
4	Hard Core Generation and Manipulation	107
4.1	Problem Statement and Goals	109

4.2	Commercial Solutions	110
4.2.1	Modular Design and BitGen	110
4.2.2	JBits	112
4.2.3	XPART	113
4.2.4	Plan Ahead	113
4.3	Academic Solutions	114
4.3.1	JBits Based Solutions	114
4.3.2	Direct Bitstream Manipulation	115
4.4	Analysis of Related Solutions and Proposal	117
4.5	Virtex II/Pro FPGAs Bitstream Format	118
4.5.1	Study Method	118
4.5.2	High Level Analysis	118
4.5.3	Low Level Analysis	119
4.5.4	Virtex II/Pro Bitstream Format	120
4.5.4.1	Bitstream Structure	120
4.5.4.2	Frame Addressing	121
4.5.4.3	CLB Frames Format	121
4.5.4.4	IOBs Frame Format	123
4.5.4.5	BRAM Content Frame Format	123
4.5.4.6	Switchboxes Analysis	124
4.5.5	Bitstream Access Equations	124
4.6	Direct Bitstream Manipulation Solution for Virtex II/Pro FPGAs	125
4.6.1	BITstream POSitioner (BITPOS)	125
4.6.2	partial BITstream POSitioner (pBITPOS)	128
4.7	Design Flows	130
4.7.1	Hard Core Design Flow	130
4.7.2	Small Granularity Reconfiguration	132
4.8	Results	132
4.8.1	Hard Core Benchmark Library	132
4.8.2	BITPOS Execution Time Results	133
4.8.3	pBITPOS Execution Time Results	136
4.8.4	Comparison with other State of the Art Solutions	142
4.9	Conclusion	145
5	Application Examples of Partial Runtime Reconfigurable Systems	147

5.1	Introduction	149
5.2	Reconfigurable Systems for Wireless Sensor Network	150
5.2.1	Wireless Sensor Network Node - Cookie	151
5.2.2	Node Reconfigurability	151
5.2.2.1	Target Use Cases Scenarios	152
5.2.2.2	Virtual Architecture Design	153
5.2.2.3	Reconfigurability Control Software	154
5.2.2.4	Platform Hard Cores	155
5.2.2.5	Reconfiguration Process Work Flow	155
5.2.3	Reconfiguration Evaluation Parameters	156
5.2.4	Test and Results	158
5.2.4.1	Transmission, Reception and Retransmission Data Rates	158
5.2.4.2	Cookie Cost Parameters	159
5.2.4.3	Transmission/Retransmission Energy and Cost	159
5.2.4.4	FPGA Configuration and Memory Cost	160
5.2.4.5	Hard Cores Area Requirements	161
5.2.4.6	Reconfigurable System Validation	162
5.2.4.7	Results Evaluation	163
5.3	Remote Reconfigurable System: ENAMORADO	164
5.3.1	ENAMORADO Environment	164
5.3.2	ENAMORADO Reconfigurable Client	166
5.3.2.1	Virtual Architecture	167
5.3.2.2	Client Device Reconfiguration Management	168
5.3.2.3	Reconfiguration Client-Server flow	170
5.3.3	Tests and Results	172
5.3.3.1	Reconfigurable System Evaluation	173
5.3.3.2	Reconfigurable Client-Server Flow Evaluation	173
5.3.3.3	Overall ENAMORADO System Evaluation	174
5.4	Reconfigurable Systems Debug	175
5.4.1	Debug System	175
5.4.2	Tests and Results	177
5.5	On-chip Communications Emulation Framework: DRNoC EMUL	179
5.5.1	Work Flow - Design Space Exploration	181
5.5.2	DRNoC Emulation Framework	182
5.5.3	DRNoC Emulation SW	184

5.5.4	DRNoC Use Cases and Results	186
5.5.4.1	DRNoCs Design Resources Cost Evaluation	187
5.5.4.2	Use Case 1: Inter-Core Reconfiguration to Change the Communication Scheme	191
5.5.4.3	Use Case 2: Inter-Core Reconfiguration of NoC Physical Layer	196
5.5.4.4	Use Case 3: Intra-Core Reconfiguration to Change NoC Parameters	200
5.5.5	Evaluation of the Proposed Emulation System	206
5.5.5.1	Speedup Analysis	206
5.5.5.2	DRNoC Emulation: Advantages and Disadvantages	208
5.6	Conclusions	208
6	Overall Conclusions and Future Work	211
6.1	Summary and General Conclusions	213
6.2	Summary of Original Contributions	216
6.3	Main Publications	217
6.4	Thesis Achievements	218
6.5	Future Work	219
6.6	Discussion on Reconfigurable Systems	220
	Bibliography	223
	Thesis Publications	237
6.7	Book Chapters and Journals	237
6.8	International Conferences	237
6.9	National Conferences with Review Process	239
	List of Symbols	241

List of Figures

1.1	On the left, ASIC and FPGA started projects statistics and on the right, FPGA market growth forecast (graphics are taken from [ins06] and [Ele08]).	4
1.2	Reconfigurable Systems in Research. Tables are taken from [Har06b]. . . .	5
1.3	Reconfigurable system microprocessor and reconfigurable fabric coupling types [AH06].	8
1.4	Virtex II FPGAs family architecture overview [Xil07].	10
1.5	Virtex II FPGAs family CLB architecture [Xil07].	11
2.1	One dimensional partially reconfigurable systems evolution time diagram.	27
2.2	Example reconfigurable system with two reconfigurable modules, defined using the Xilinx recommendations [DL04].	29
2.3	Xilinx reconfigurable system on-chip communication structure - BM-TBUF-Xilinx. Communication structure position on the left side and a detailed view on the right side.	30
2.4	Lockwood-Horta et al. On the left side, the Reconfigurable Application Device (RAD) with two reconfigurable modules called Dynamic Hardware Plugins (DHP). On the right side, a detailed view of the DHP module fixed interfacing signals called "antennas" [ELH02].	31
2.5	Moraes-Calazans et al. reconfigurable system. On the left side the 1D architecture general view and a detailed view of the on-chip communication serial bus on the right side [PdMM ⁺ 02].	33
2.6	Platzner - Walder et al. reconfigurable system, oriented to provide HW support to operating systems (OS). Example with five reconfigurable slots and two OS frame on the FPGA corners and an example of loading different slots of different width [WP04].	34
2.7	Reconfigurable system example with four reconfigurable modules [WP04].	35

2.8	Hübner-Becker et al. CLB bus macro (BM-Hübner) layout. Screen shot from the FPGA Editor tool [MUB04].	36
2.9	Hübner et al. bus macros with highlighted bus accesses. On the top part, the input macro that sends data from slots to the fixed area, and on the bottom part the output macro that sends data from the fixed area to slots [MUB04].	36
2.10	Erlangen Slot Machine (ESM) architecture. The Erlangen-Nürnberg reconfigurable systems design platform is composed of two boards. The MotherBoard and the BabyBoar, which provides partial reconfigurability. In the shown example, the system contains three reconfigurable modules and all the existing modules connectivity [MABT06].	37
2.11	Virtual architectures for partial runtime reconfigurable systems - design method general steps.	40
2.12	FPGA Editor layout. For a Virtex II FPGA on the left and for a Virtex II Pro on the right.	43
2.13	Resource Division for Virtex II FPGAs on fixed and reconfigurable area on the left, and for Virtex II Pro on the right.	43
2.14	One dimensional general virtual architecture for Virtex II FPGA families on the left and for Virtex II Pro on the right.	46
2.15	Two dimensional virtual architecture general view on the left and an example mapping for Virtex II on the right.	46
2.16	TBUFs and TBUFs Routing Resources [Xil04].	50
2.17	Routing resources available in Virtex II/Pro FPGAs [Xil04].	51
2.18	Point to Point Interconnection (P2P) example. A short interconnection is shown on the top part of the figure and a long interconnection is shown in the bottom part.	52
2.19	Two horizontal CLB based communication macros interconnection (BM-LUTs) FPGA Editor images. One short macro that communicates two neighboring elements (BM-LUT-1) is shown on the top part and on the bottom, a long macro where one of the interconnection points stands at six CLBs distance (BM-LUT-6).	53
2.20	Bus communication structure for an XC2V1000 (Bus-v1) built by long CLBs and buffer based macros.	57
2.21	Bus communication structure for an XC2V3000 (Bus-v2), built by long CLBs and buffer based macros.	58
3.1	The OSI protocol stack, the Luca Benini and Giovanni de Michelli stack for NoCs proposal, called micro-network stack and the stack proposed by Dehyadgari et al.	65
3.2	NoCs for reconfigurable systems research topic - time evolution. The graphic includes all the research groups described in this Chapter, along with the proposed solution.	67

3.3	Nollet-Mignollet et al. proposal. Architecture general view on the left. Virtex II FPGA implementation on the right [NMV ⁺ 04].	68
3.4	Hübner-Becker et al. 2D system, focused on-chip connections adaptation based on partial reconfiguration. The system permits a core to be connected/disconnected to a communication structure. On the left, the architecture general view and on the right, the connection reconfiguration resources [JBH].	70
3.5	The DyNoC architecture, proposed by Christophe Bobda et al. from the University of Erlangen-Nürnberg. It permits to load cores with different size. The architecture general view can be seen on the left size and on the right side the implementation on a Virtex II FPGA [BA05]. . . .	72
3.6	Piontec et al. on-chip communication solution, called CoNoChi, general view on the right side and an implementation on a Virtex II Pro on the left side. The architecture permitted reconfigurability is to load different size cores and to add new routers to the NoC.	73
3.7	Moraes-Calazans et al. ARTEMIS NoC. A general view on the left and a Virtex II implementation on the right side. This NoC permits nodes to be reconfigured.	75
3.8	DRNoC Architecture General View. Heterogeneous cores are interconnected by an Xmesh of reconfigurable routing modules. Cores access the communication network through reconfigurable network interfaces. . . .	77
3.9	A 6x6 DRNoC architecture with four independent communication schemes mapped.	78
3.10	On the left side, a general view of a 2D virtual architecture designed with the method defined in Chapter 2 and, on the left side, the DRNoC approach mapped on a slightly modified virtual architecture.	80
3.11	DRNoC mapping to two Virtex II FPGAs: a 4x4 DRNoC on an XC2V8000 on the left and a 2x2 DRNoC on an XC2V3000 on the right.	82
3.12	DRNoC addressing on the left and the packet format on the right. Three types of addresses are differentiated: Slot address, Node address and Core ID. The packet header part is divided into phit units (uphits), while the payload is divided into flits.	84
3.13	DRNoC router general view with highlighted data and control paths. . . .	86
3.14	DRNoC NoC Models Generation Tool:DRNoCGEN. The DRNoCGEN Tool virtual architecture definition screen shot is shown.	90
3.15	DRNoC RT area results for different number of ports on the left and different routing table sizes on the right.	102
3.16	DRNoC RT frequency results for different number of ports on the left and different routing table sizes on the right.	102
4.1	Virtex II/Pro full bitstream structure.	120
4.2	Virtex II/Pro bitstream format.	121

4.3	CLBs frame addressing. CLB LUTs configuration frames that correspond to frame minor addresses 1 and 2.	122
4.4	Hard core example initial and target position. The example is based on a virtual architecture with 2D slot distribution presented in Chapter 2. . . .	127
4.5	BITPOS execution window print screen image. Execution in merge mode with verbose output activated. The parsed description files options values can be seen in the figure.	128
4.6	pBITPOS re-target mode execution window print screen image. The parsed description file option values can be seen in the figure.	129
4.7	Hard core design flow. The flow starts with the selection of an architecture template, which results from the virtual architecture design method steps. The flow also uses the BITPOS tool presented in this Chapter.	131
4.8	Different hard cores file sizes, from the smallest that contains a single CLB column and a single BRAM column (01/01) to a full bitstream.	133
4.9	BITPOS execution time in merge mode for generating different hard cores for Virtex II FPGAs. Sizes go from 1 CLB column and use a single BRAM column (01/01 in x axis) to 22 CLB columns and one BRAM (22/01). . . .	134
4.10	BITPOS execution time in merge mode for generating different hard cores for Virtex II Pro FPGAs. Sizes go from 1 CLB column and use a single BRAM column (01/01 in x axis) to 18 CLB columns and 4 BRAMs (18/4). . . .	134
4.11	Detailed view of the execution time needed for an XC2V3000 FPGA for hard cores with different size that access different number BRAM columns and that do not access BRAM columns.	135
4.12	pBITPOS execution time in merge mode for Virtex II devices and for hard cores of different size. The x axis shows hard core size and number of BRAMs used by the core. Core sizes go from 1 to 22 CLB columns, while associated BRAMs go from 1 to 2.	137
4.13	pBITPOS execution time in merge mode for Virtex II Pro devices and for hard cores of different size. The x axis shows hard core size and number of BRAMs used by the core. Core sizes go from 1 to 18 CLB columns, while associated BRAMs go from 1 to 4.	137
4.14	Detailed view of the pBITPOS execution time needed for a Virtex II XC2V3000 FPGA for hard cores with different sizes that access different number BRAM columns (1 or two) and that do not access BRAM columns. Additionally, the execution time in simple mode is also included.	139
4.15	Detailed view of the pBITPOS execution time needed for a Virtex II Pro XC2VP30 FPGA for hard cores with different sizes that access different number BRAM columns (from one to four) and that do not access BRAM columns. Additionally, the execution time in simple mode is also included.	139

4.16	pBITPOS execution time in re-target mode for re-targeting three hard cores with different sizes, designed for a Virtex II XC2V500 FPGA, , to different Virtex II FPGAs. From left to right: to an XC2V1000, an XC2V3000 and an XC2V8000. As reference the execution time in normal merger mode (no re-target) of an XC2V3000 is also included (XC2V3000MM).	141
4.17	pBITPOS execution time in re-target mode for re-targeting three hard cores, designed for a Virtex II XC2V500 FPGA with different sizes to different Virtex II Pro FPGAs. From left to right: to an XC2VP20, an XC2VP30, an XC2VP50, and an XC2VP100.As reference the execution time in normal merger mode (no re-target) of an XC2V3000 is also included (XC2VP30MM).	141
4.18	Tools and Flows for Hard Core design and bitstream manipulation time diagram.	142
5.1	Cookie platform [PdCdITR06] processing layer.	152
5.2	Spartan 3 FPGA virtual architecture. The architecture is composed of three slots with different size and specific functionality. The FPGA communicates on the left side with the Cookie uC and on the right side with the Cookie digital sensors. The on-chip communication is based on CLB neighboring connection macros.	153
5.3	Reconfiguration Process Work Flow Diagram.	156
5.4	System use case application FPGA Editor screen-shots. Analog sensors node configuration (AS_HW_AVRF) on the left side and the digital sensors configuration example (DS_HW_AVRF_TMPIF) on the right side	162
5.5	ENAMORADO (Enabling Nomadic Agents in a Multimedia ORiented Architecture of Distributed Objects) environment general view.	165
5.6	1D bus architecture for the target XC2V3000 FPGA (bus-v2), along with the related FPGA architecture definition string in the upper part.	167
5.7	FPGA reconfiguration control middleware.	169
5.8	Device profile - reconfigurable system related fields. The shown fields are part of the extension applied to the MPEG21 standard in order to provide reconfigurability support.	170
5.9	ENAMORADO reconfigurable client device composed of a commercial tablet PC platform (XINGU) with an XScale processor, shown on the right side, and a custom board specially designed for partially reconfigurable systems prototype shown on the left side.	172
5.10	Using partial reconfiguration for debug.	175
5.11	CHDT execution screen shot while dealing with a hard core debug module.	177
5.12	DRNoC design space exploration work flow. The main, distinguished characteristic of the flow is that it is based on hard cores re-usability. Notice, that some steps of the flow are manual.	181

5.13	DRNoC emulation framework, divided in three platforms.	183
5.14	DRNoC Emulation SW. The SW is in charge of controlling DRNoC configurations and generating appropriate FPGA bitstream files. The tool is based on the work flow for design space exploration using partial reconfiguration, proposed in Chapter 4 and The BITPOS and pBITPOS tools.	185
5.15	Picture of the Emulation Framework boards. XUP board on the left, a DRNoC mapped on the proprietary development board on the right. The JTAG parallel cable III can be also seen in the picture.	186
5.16	Test of Inter-Core reconfiguration used to change the communication scheme. The application CTG is shown on the top left edge of the Figure. Its DRNoC based model is shown on the top right corner and two configurations are defined and shown in the bottom part: one is a pure NoC (CS1) on the left and the other one is a pure point two point (P2P) on the right (CS2).	191
5.17	FPGA Editor floorplanning for CS1 (NoC) on the left and CS2 (P2P) on the right.	193
5.18	CS1 (NoC) current latency and current throughput, measured at core level. The burst behavior can be clearly seen in both curves.	194
5.19	CS1 (NoC) current latency and current throughput, measured at NI level. The burst behavior can be seen in both curves.	195
5.20	CS2 (P2P) current latency and current throughput, measured at core level. The burst behavior can be seen in the throughput, while the latency is constant.	195
5.21	Test of Inter-Core reconfiguration to change a communication scheme physical layer. The application CTG is shown on the top part of the Figure, along with its DRNoC mapping. Two communication schemes configurations for this application can be seen on the bottom part of the Figure: one Mesh NoC (CS1 on the left side) and one Star NoC (CS2 on the right side).	196
5.22	CS1 (mesh NoC) current latency and current throughput plot, measured at core level.	198
5.23	CS1 (mesh NoC) current latency and current throughput plot, measured at NI level.	199
5.24	CS2 (star NoC) current latency and current throughput plot, measured at core level.	199
5.25	CS2 (star NoC) current latency and current throughput plot, measured at NI level.	200

5.26	Test of Intra-Core reconfiguration used for changing a NoC parameter. An application CTG and the DRNoC model, made up by four different TG-TRs and one measurement point in node3, can be seen on the top part of the Figure. The application has two configurations, shown on the bottom part. Both have the same communication scheme (Mesh NoC), but different buffer size: one with a FIFO of 64 entries (CS1) and another with a FIFO of 256 entries (CS2).	201
5.27	CS1 (64 entries FIFO) and CS2 (256 entries FIFO) NoC latency characterization.	203
5.28	CS1 (64 entries FIFO) current Latency and injection rate plot, measured at core level.	204
5.29	CS1 (64 entries FIFO) current Latency and injection rate plot, measured at NI level.	204
5.30	CS2 (256 entries FIFO) current Latency and injection rate plot, measured at core level.	205
5.31	CS2 (256 entries FIFO) current Latency and injection rate plot, measured at NI level.	205
5.32	Emulation system evaluation for each use case presented in this section. Speedup comparison of the proposed, partial reconfiguration based approach, with respect to: i) simulation (Total emulation to simulation), ii) a full synthesis based approach (PR synthesis to synthesis) and iii) speedup of the worst case core synthesis time with respect to the full system synthesis time (PR emulation to synthesis).	206

List of Tables

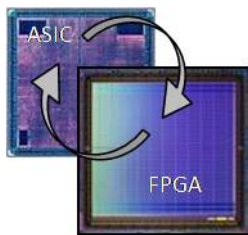
2.1	FPGA Virtual Architecture Definition Parameters	45
2.2	Virtual architectures design relations for Virtex II FPGAs.	47
2.3	Virtual architectures design relations for Virtex II Pro FPGAs.	47
2.4	Virtual architecture slots distributions for Virtex II - 1D Model	48
2.5	Virtual architecture slots distributions for Virtex II - 2D Model	48
2.6	Virtual architecture slots distributions for Virtex II Pro - 1D Model	49
2.7	Virtual architecture slots distributions for Virtex II Pro - 2D Model	49
2.8	P2P buffer and CLB based communication macros features comparison . .	54
2.9	Systems template pseudo code	55
2.10	Hard core template pseudo code	55
2.11	bus based 1D reconfigurable systems - features summary. Overall comparison of all the approaches presented in the state of the art section of the Chapter and other, based on the proposed in this thesis method. . .	60
2.12	Bus communication infrastructures comparison	61
3.1	Two example DRNoC implementations on Xilinx Virtex II FPGAs VA. . .	83
3.2	Area overhead and percentage use of the target slot (slot/core) for several Traffic Generators (TGs) and regular traffic generators (TG-MX-Regular) for different amount of target Traffic Receivers (TRs).	96
3.3	Area overhead and use percentage of the target slot (slot/core) for two types of Traffic Receivers (TRs) and online TRs (TR-MX-online) that include different number of measuring points.	97
3.4	Area overhead and use percentage of the target slot (RNI) for several types Network Interfaces (NIs) for the receiver's side and NIs for and 8 bit flit NoC that include different number of online measuring points (NI-TR-MX-online-8b).	97

3.5	Area overhead and use percentage of the target slot (RNI) for traffic generators Network Interfaces (NI-TG) for two types of NoCs, 8 bit flit and 16 bit flit.	98
3.6	Area overhead and use percentage of the target slot (RRM) for an 8 bit flit XY NoC routers (R-XY) for different number of used communication channels, and a two channels router for a 16 bit flit NoC.	98
3.7	Two dimensional reconfigurable systems with Network on-Chip connectivity comparison table. The comparison is based on structural parameters defined in section 3.6.2.	99
3.8	HERMES and DRNoC Routers area requirements	101
3.9	Some state of the art and DRNoC routers required FPGA area comparison for seep and area optimization.	101
3.10	NoC based reconfigurable systems features summary. Summary of all the approaches described in introduction section of the Chapter and the originally proposed DRNoC solution.	103
3.11	NoC based reconfigurable systems. Implementations and applications related features.	104
4.1	JBits pseudo code	112
4.2	CLB frame format	122
4.3	CLB Configuration Bits	123
4.4	BRAM content frame format	123
4.5	Definitions	124
4.6	Description (desc) file values. These files are used to define the core initial and target position. Some of the values included in the example are not obligatory, like the initial position.	126
4.7	Hard cores generation and manipulation tools features summary	144
5.1	Data transmission time and rate for the JTAG configuration SW, and for a hard core that corresponds to slot2 - 4 columns wide.	158
5.2	Transmission/reception (both values are equal) energy and cost parameters.	159
5.3	Retransmission energy and cost parameters	159
5.4	Hard cores memory cost	160
5.5	SW programs memory cost	160
5.6	Reconfiguration cost Parameters	161
5.7	Hard Cores area requirements	161
5.8	Use cases hard cores description.	188
5.9	Use cases hard cores cost parameters.	189

5.10 Slots reconfiguration cost and time for Intra and Inter-Core reconfigurations.	190
5.11 Total cost parameters for two communication schemes: one P2P (CS1) and one NoC (CS2) for the pipeline application (Use Case 1-UC1).	193
5.12 Maximum and minimum latency and throughput values for two communication structure CS1(NoC) and CS2 (P2P) configuration for UC1.	194
5.13 Total cost parameters for the mesh NoC (CS1) and the star NoC (CS2) of UC2.	197
5.14 Max. and min. latency and throughput for CS1(mesh NoC) and CS2 (star NoC).	198
5.15 Total cost parameters for the two mesh NoC communication strategies (CS1 and CS2) related to the application used in UC3.	202
5.16 Max./min. current latency and current injection rate values for two communication structures, CS1(64 entries FIFO) and CS2 (256 entries FIFO), for the UC3 application.	203
5.17 Proposed partial reconfiguration base emulation framework advantages and disadvantages	208

Chapter 1

Introduction



Introduction to the Field Programmable Gate Array (FPGA) technology framework and reconfigurable computing based on FPGAs. General overview of commercial and academic reconfigurable solutions. Partially reconfigurable systems based on commercial FPGAs, design considerations and thesis approach.

This Chapter is an introduction to the topic of reconfigurable computing based on FPGAs and more specifically in the thesis core issue, partial reconfiguration. The Chapter also justifies some of the thesis general approaches and discusses several aspects of the design of flexible reconfigurable systems that have been targeted. The Chapter begins, in section 1.1, with an FPGA technology framework overview, where the importance of the reconfigurable computing in the electronic industry is introduced. After that, some outstanding academic and commercial reconfigurable devices are described in section 1.2. Section 1.3 discusses the main topic of the thesis work, the partial reconfiguration technology, highlighting its advantages and introducing some concepts to be managed along the document. Section 1.4 presents a deeper view into the topic of partial reconfiguration based on commercial FPGAs with a brief state of the art. Section 1.5 introduces some of the approaches to be followed in this thesis and finally, a short summary and the thesis objectives can be found in section 1.6.

1.1 Technology Framework

There is almost no doubt reconfigurable computing is gaining more importance in the electronic industry. This section describes the place that is occupied by FPGAs, as main reconfigurable systems representative, in the electronic market and the technology.

Traditionally, the electronic industry has selected FPGAs as device prototyping platforms, but afterwards, Application Specific Integrated Circuits (ASICs) were preferred for the final product.

ASICs are preferred for large and serial device productions, where the cost per unit is reduced and the high cost production masks are amortized. ASIC implementations provide high performance and low power consumption, but they have an extensive design and large time to market, typically one to two years for a standard cell ASIC design and more for full custom devices. Furthermore, the current electronics productions paradigm has caused an exponential growth of manufacturing non-recurring engineering (NRE) cost, where mask inspection and repair is the largest component of cost and product delay [ITR03]. To partly solve this problem and reduce the price, structural ASICs provide a quick turn ASIC solution, with lower NRE cost, and equivalent power and performance.

On the contrary, FPGAs are preferred in final products with low serial production volumes and relatively low production cost, where time-to-market is highly important and the lowest non-recurring engineering is desirable.

From this scenario it seems that ASICs and FPGAs have their own market place. But during the last years, FPGAs are putting ASICs aside, entering into their traditional market. The ASICs market is being reduced, less ASIC projects have been started, as it can be noticed from the diagram included on the left side of Figure 1.1, provided by Nikkei Electronics. Differently, the design and production of FPGAs is constantly increasing [Ele08], as it can be noticed from the FPGA design curve, included in the left side of Figure 1.1). The market is searching for more "reconfigurable" solutions leading to the inclusion of more and more FPGAs in final products. According to In-Stat, leading company in market analysis [ins06], the FPGA market will grow from 1,9 M \$ in 2005, to 2,75 \$ in 2010, as it can be seen on the diagram shown on the right side of Figure 1.1, being the United States and Asia the major consumers. The same company is also forecasting that telecommunication industry, one of the main applications of FPGAs, will rise from 73,8% in 2005 to 76,8% in 2010. Other sources, like Actel (FPGA vendor), forecast that the FPGA market will exceed the 6,2 M \$ in 2010 and the major FPGA provider, Xilinx, is forecasting 5-9 percent growth in the financial year of 2009, which started in April 2008.

Numbers apart, the tendency in the growing FPGA market is clear. These changes have resulted from the transformation of classic configurable gate arrays in complex reconfigurable platforms, composed of a broad set of reconfigurable elements [lib05] which can be reprogrammed using several methods. The FPGA integration level currently reaches a few million system gates and several elements are embedded on the silicon die; microprocessors, memories and Digital Signal Processing (DSP) specific cores, like multipliers and DSP slices. Furthermore, an important factor is the constantly increasing number of Input Outputs (IOs) available in the device, the supported

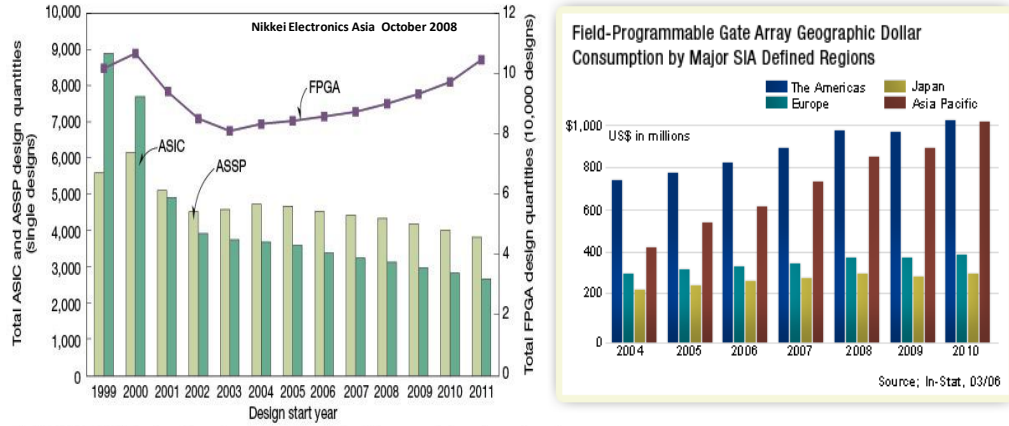


Figure 1.1: On the left, ASIC and FPGA started projects statistics and on the right, FPGA market growth forecast (graphics are taken from [ins06] and [Ele08]).

interfacing standards and the large number of available Intellectual Property Cores (IP cores). As a result, more FPGAs are integrated in products, making possible the final device to be the own prototype, *drastically reducing time-to-market* and production costs, resulting in highly flexible reconfigurable systems, with *larger time-in-market* and cheaper hardware maintenance as they can be dynamically updated after system deployment.

Furthermore, it is known that a growing portion of the design time is spent on dealing with deep submicron effects. Surveys show that, today, less than 20% of the design time, for complex Systems on-Chip (SoCs), is spent on design [Bol03]. By using FPGAs, system designers can focus their work on the entire system as production dependent effects have already been overcome by FPGA providers.

In a different aspect, FPGAs, through their regular and inherently parallel architecture are a possible alternative to the Von Neumann architectures that have been dictating sequential programming models [Har07]. FPGAs permit to effectively run the massive data treatment algorithms of the complex applications of today and future consumer and industry electronics.

Even more, now, FPGA providers directly attack the ASIC market and have started offering a solution to pass directly from FPGAs to structural ASICs, depending on the production evolutions. Altera, the second major FPGA provider calls this solution, "HardCopy ASIC" while Xilinx, the major FPGA provider, has its "EasyPath FPGAs". This way, the former FPGA volume production limits are being blurred and design changes can be implemented using the same FPGA until the final working solution or production volumes are reached, and then it is replaced with a pin compatible ASIC device.

From the point of view of the research opportunities, reconfigurable computing (based and not based on FPGAs) has had an important growth in the last years. This growth can be noticed by looking at the increased number of publications. In Figure 1.2, the number of conferences and congresses in the reconfigurable computing topic and the number of papers submitted to the most outstanding FPGA conferences [Har06b], like the FPL (Field Programmable Logic) are shown. In this line, Europe plays a leading role, the FPL annual conference series is the oldest and by far the largest in the world [Har06a]. Furthermore, in the academic area, FPGAs are enhancing their application domain. It is being more and more common to find them applied to a variety of electronic systems [MSea97], [Bou06] and [RAMV07]: from industrial control solutions [RSMC07], to complex and high performance systems[RSMC08]. Now it is possible to implement a complete neural network and fuzzy control based solution on FPGAs [CYS06], something that was not possible a few years ago.

category of international conference series	listed	FPL		
directly on Reconfigurable Computing	26	year	submissions	attendees
Biology-inspired Reconfigurable Computing	2	2006	390	
having RC as a main point of focus	60	2005	364	~300
having special tracks or keynotes on RC	33	2004	289	~300
RC in Supercomputing or Computer Conf'es	34	2003	~250	~270
total listed in this memo:	155			

Figure 1.2: Reconfigurable Systems in Research. Tables are taken from [Har06b].

It is important to remark that reconfigurable computing is not restricted to FPGAs. However, *this thesis is focused on commercial devices in order to try to keep the solutions proposed along the document closer to the industry. At the time this document is being written, the most widely used option for dynamically reconfigurable systems are Xilinx FPGAs.* More specifically, this thesis research is focused on partial RunTime Reconfigurable Systems (pRTRS) that powers the benefits of using reconfigurable systems by allowing updates in running systems. This subject, among others, will be introduced in the next section and after that, in the following sections, the most outstanding commercial reconfigurable devices, as well as some custom reconfigurable systems will be described in detail.

1.2 Reconfiguration Techniques and Systems

This section reviews the types of reconfiguration techniques, granularity, reconfiguration rate and system coupling that can be found nowadays. These parameters are closely related to system data transfers, latency, power, throughput and overhead that define the system performance.

A reconfigurable device is configured with a sequence of bits organized in a file called bitstream. Some years ago, FPGAs had to be configured with a full configuration file in a static configuration process. By opposite, nowadays several reconfiguration techniques, explained next, are available:

- **Full Reconfiguration.** This type of reconfiguration is also called global or total reconfiguration. Dynamic, full, reconfiguration is a device programming method, which allows configuring the entire device without the need of resetting the remaining external circuitry. The main advantage of this method is that the reconfiguration process is well known, stable and is supported by most of the FPGAs available nowadays. On the other hand, one disadvantage is that configuration files are not only quite large (in the Megabytes range) but also their size is constantly increasing. Consequently, as entire bitstreams have to be loaded in the FPGA, the time needed for a full reconfiguration is long (several seconds) and the memory required to save these files is large. Additionally, it is well known that FPGAs power consumption increases during reconfiguration, so spending more time in reconfiguration leads to an increase total power consumption. Anyhow, the main disadvantage of full reconfiguration is the limited system adaptability. The device input/outputs cannot be used during reconfiguration and this affects the function of the entire system and even more, the entire device has to be reserved to a single application.
- **Partial Reconfiguration** is a type of reconfiguration which allows to modify only a piece of a device. Partial reconfiguration can be:
 - i) non disruptive, runtime or also called dynamic, where the portions of the system, which are not being reconfigured, remain fully operational during the reconfiguration cycle and
 - ii) disruptive or non dynamic where partial reconfiguration affects other portions of the system that typically need for a clock hold, or inputs/outputs to be kept at high impedance. The main advantages of the partial reconfiguration technique are: i) reduced programming time that ranges from several ms. to a few seconds, ii) high silicon reuse (the FPGA area can be used by different tasks running in parallel) and iii) reduced power consumption and memory requirements (due to smaller bitstream files). On the other hand, the main disadvantages are the lack of suitable tools and design methods. Actual design methods do not permit to design flexible systems and require specialized designers. Another disadvantage is the increased system complexity and the need for a specialized reconfiguration control system. These topics will be further and deeper discussed in section 1.5.
- **Self Reconfiguration.** In this reconfiguration method the reconfigurable device controls its own reconfiguration process. Self reconfiguration requires partial reconfiguration, but is usually referred as a separate technique.

As it has been already mentioned, this thesis work is focused on partial runtime, dynamic reconfiguration. From here after, partial runtime dynamic reconfiguration will be referred also as dynamic reconfiguration or simply as reconfiguration and, when other types of reconfiguration are referred they will be specified (full or self reconfiguration). Self reconfiguration is addressed in one of the applications presented in Chapter 5 (the application Chapter).

Reconfiguration technology involves granularity definition. Granularity is defined as the size of the smallest functional unit that can be modified [BP02]. In this aspect, reconfigurable systems can be classified in several groups listed below:

- *Coarse granularity* refers to the modification or reconfiguration of complete IP cores.
- *Medium granularity* is when dealing with part(s) of a core and is performed at register level.
- *Low granularity* also called fine grain is when a single or several bits are modified.

Lower granularity permits more flexibility in adapting the system hardware, but requires more time to construct complete computational modules [BP02]. Coarse granularity, on the other side, permits faster system construction, but is less flexible.

The proposals presented in this thesis, as it will be seen, are mainly focused on coarse grain reconfigurable systems. Reconfigurable systems, designed along the thesis, mostly deal with entire cores. However, the method presented in Chapter 2 also covers the remaining granularity levels that are exploited in some cases in order to increase the system flexibility.

A typical reconfigurable system is composed of a microprocessor and a reconfigurable fabric. The microprocessor is in charge of the system and/or peripherals control, while the FPGA runs computationally hungry tasks. In this way, the system takes advantage of the software flexibility and the hardware parallelism.

There are several coupling schemes for such systems, graphically presented in Figure 1.3 and briefly described below [AH06] and [CH02]:

1. *External stand-alone.* The reconfigurable hardware is a fully independent device connected to the microprocessor inputs and outputs to provide adaptable functional units in a host processor system. In this case reconfigurable units execute as functional units on the main microprocessor data path and registers are used to hold the input and output operands.
2. *Attached processing unit.* The attached processing unit or coprocessor in general is larger than a functional unit and can perform computation without the constant supervision of the host processor. This permits the reconfigurable and the host processor to operate independently and simultaneously, reducing the overhead.
3. *Coprocessor.* This coupling type is somewhere in between the first two types. In this situation, reconfigurable logic is embedded in the data cache that can be used either as regular cache or as additional computing resource.
4. *Reconfigurable functional unit.* In tightly coupled systems, the functional unit is part of the microprocessor functional units and might permit to define custom instructions.
5. The last type is when the *microprocessor is embedded in the FPGA fabric*. In this type, the processor could be a hard core melt into the silicon die, or it could be a soft core processor implemented with FPGA logic. It could be argued that this type of coupling could be the second or the third type, depending on the way the platform is being configured and therefore it is defined in a separate type.

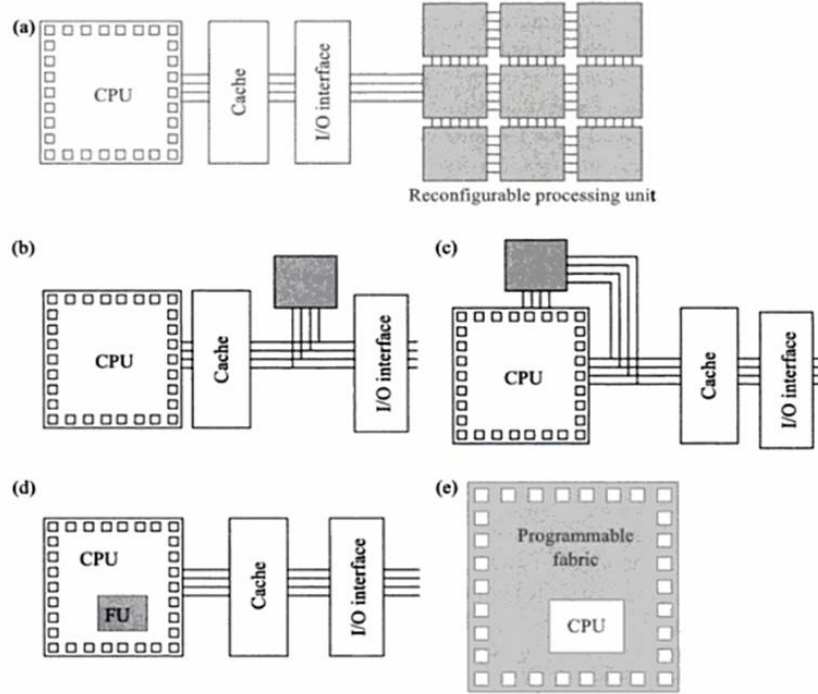


Figure 1.3: Reconfigurable system microprocessor and reconfigurable fabric coupling types [AH06].

Related to the type of cores that a system can manage, IP cores can be classified into three groups, depending on their abstraction level and flexibility [GZ97]. The groups, introduced next are: soft, firm and hard cores.

1. **Soft cores** are technology independent synthesizable HDL descriptions that could be modified by designers and therefore have the highest flexibility level.
2. **Firm cores** are technology independent netlists, mainly in EDIF format, ready for mapping, placement and routing with some customizable parameters and middle flexibility level.
3. In reconfigurable systems, **hard cores** are already placed and routed designs ready to be loaded in a system and thus are the less flexible ones. It is important to mention that cores melted in silicon are also referred as hard cores, but when dealing with dynamic reconfigurable systems, hard cores are represented by configuration bitstream files.

Another parameter, used to differentiate reconfigurable systems is the reconfiguration rate of a system. This parameter defines how often the system has to be updated. In this aspect, it could be differentiated:

- **Context switch.** A system is reconfigured to update a certain system aspect or to change the application context. This is the lowest reconfiguration rate (once per day, month or year(s)). In this case, system update delays are not important.
- **Runtime.** In this type, the system updates can be performed right before the execution of an application task, while others are running in parallel (multi-context) or, during the own application task execution (single-context). The reconfiguration rate has to be kept low and does not have to affect the running system functions, but the reconfiguration process is not fully transparent.
- **Realtime.** In this case, the system is reconfigured not only in runtime and in such a way that the application execution is not affected, but the system reconfiguration delay (preparing and loading the configurations) is masked and the reconfiguration process is fully transparent. This type of reconfiguration brings the software flexibility to hardware, but is highly difficult to be supported with the currently available reconfiguration technology.

This thesis is focused on runtime reconfiguration. The tool for configurations preparation (described in Chapter 3) intent to provide multi-context, runtime, switching to the second application described in Chapter 5.

Once the concepts to be managed in the thesis have been defined, in the next section, some relevant state of the art commercial and academic reconfigurable devices are presented.

1.3 Reconfigurable Devices

This section will review some outstanding dynamic reconfigurable devices from the commercial and academic area.

1.3.1 Commercial Reconfigurable Devices

At the time this work is being written, the leaders in the FPGA market are Xilinx with more than 50 % of the market share, Altera with around 30 % and then Actel and Lattice. All other FPGA providers, like Quckilogic and Atmel share the remaining market slice. A representative FPGA device, from the biggest FPGA provider, Xilinx, will be described in the next subsection. Afterwards, FPGA providers' strategies of smaller market players, Altera, Atmel and Lattice, will be briefly described.

1.3.1.1 Xilinx

In the 80's, Xilinx was the first semiconductor company to establish the fabless production model and are still leaders in the programmable logic market [Xil]. Xilinx FPGA devices have two main series: high performance Virtex series, and the low cost Spartan series.

The first families in the Virtex series were the Virtex and VirtexE FPGAs. In these families, apart from the configurable logic, used to implement users' logic, there are also RAM memory blocks, organized in columns and distributed along the device. The VirtexE family variation has enhanced memory compared to Virtex.

The second Virtex families, launched by Xilinx, were the Virtex II and Virtex II Pro FPGAs. Their internal structure is described in detail in the following paragraphs as they are the target platforms in this thesis work.

The Virtex-II family is a platform FPGA developed for high performance that is based on an 8-layer metal CMOS process and is optimized for high speed. The Virtex II family comprises 11 members, ranging from 40K to 10M system gates. An overview of the Virtex II architecture is shown in Figure 1.4. It is basically composed by an array of internal configurable logic surrounded by IOBs.

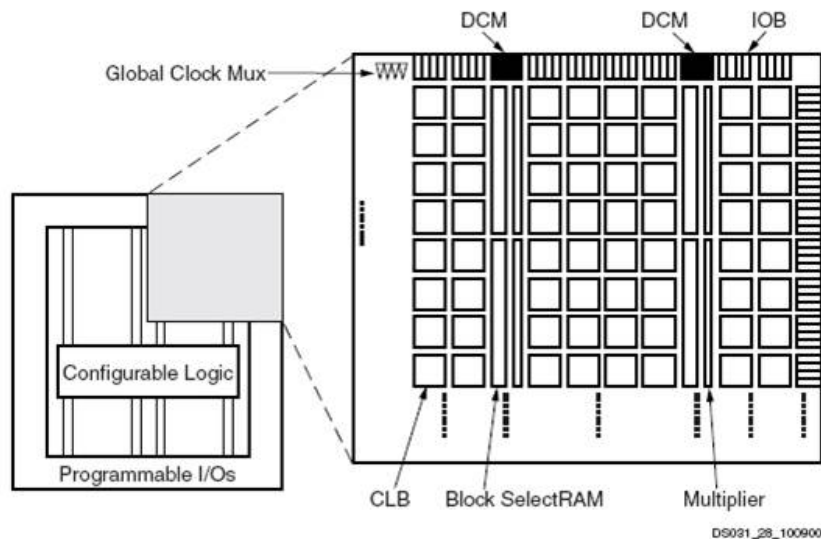


Figure 1.4: Virtex II FPGAs family architecture overview [Xil07].

The internal configurable logic includes four major elements [Xil07]:

1. **Configurable Logic Blocks (CLBs)** provide functional elements for combinatorial and synchronous logic implementation and, also, for distributed basic storage elements. A Virtex II CLB architecture is shown in Figure 1.5. A CLB is composed of four slices arranged in two columns of two slices, and two 3-state buffers (TBUFs) associated with each CLB element that drive dedicated horizontal routing resources as shown in Figure 1.5. A slice is composed of two 4-input **Look Up Tables (LUTs)** and D-type storage elements. A LUT can be configured to implement any logic function of up to 4 inputs, as a single- or dual-port RAM, or a shift register. The storage element can also be configured as a flip-flop or a latch. Additionally, each slice includes fast carry logic and direct connection with other LUTs without using the switch matrices.

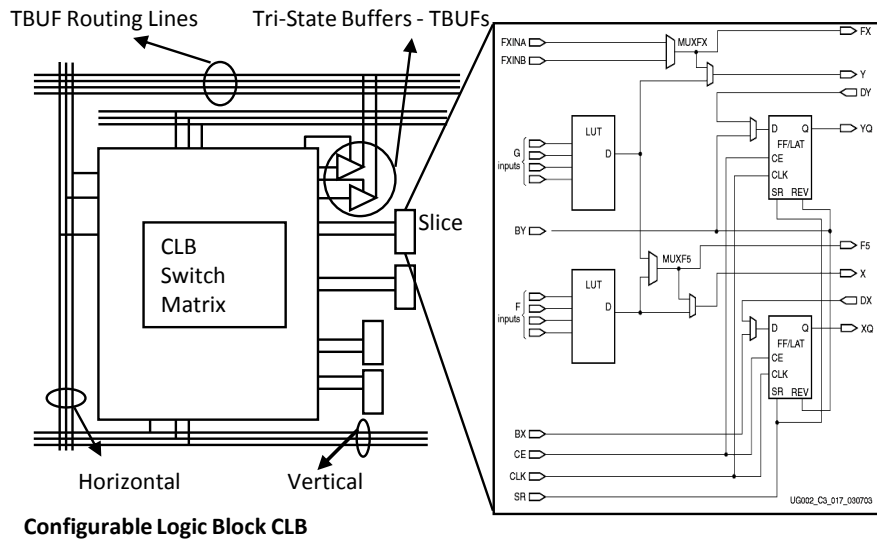


Figure 1.5: Virtex II FPGAs family CLB architecture [Xil07].

2. **Block SelectRAM (BRAM)** memory modules, which provide 18 Kbit storage elements of dual-port RAM and are organized into BRAM columns that are interleaved in the array of CLBs (see Figure 1.4).
3. **Multiplier blocks (MULs)** are 18x18 bit dedicated multipliers. There is a MUL column right next to a BRAM one. Therefore when BRAM columns are referred, along the thesis, MULs will be also taken into account.
4. **Digital Clock Manager (DCM)** blocks provide self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division, coarse- and fine-grained clock phase shifting.

In Xilinx FPGAs, all configurable elements are interconnected by programmable routing resources based on a proprietary technology called "active interconnect technology". Each programmable element is tied to a Switch Matrix (SM) and all switch matrices are organized in an array called general routing matrix.

The latest Virtex families are the Virtex 4 and Virtex 5. With Virtex 4 Xilinx has enhanced the variety of devices in a family and the initial Virtex 4 family includes three platforms: Virtex 4 LX with high number of logic cells, Virtex 4 SX for very high performance signal processing, and Virtex 4 FX for embedded processing and high-speed serial connectivity. The Virtex 4 CLB structure is quite similar to that of the Virtex II, each CLB contains four slices and uses four input LUTs [Xil08a]. The logic resources of Virtex 4 go from 13M to 200M logic cells. Differently, with Virtex 5 FPGAs, Xilinx has introduced a coarser architecture moving to six input LUTs. Each CLB in Virtex 5 is composed of half the amount of slices, two, compared to previous FPGA generations, but in Virtex 5 each slice has twice the amount, a total of four, of LUTs and registers. This results in the

same amount of LUTs per CLB as previous architectures, but increases the equivalent logic resources that go from 30M to 330M logic cells. Apart from this, Virtex 5 FPGAs have enhanced storage capabilities (36-Kbit BRAM) and also include special DSP slices organized, like BRAMs, in columns [Xil08b].

Regarding the low cost Spartan series, its architecture is very similar to the Virtex II (CLBs structure). The main difference with the high performance counterpart is the reduced amount of logic resources, clock frequency and per unit cost. There are several families in this series: The first device family was the Spartan 3 with equivalent logic cells from 1M to 73M. Then, the Spartan 3E family with increased amount of logic per I/O and another with enhanced power features Spartan 3A (suspend and hibernate power modes) were released. The latest Spartan 3 FPGAs families are nonvolatile and include the Spartan 3AN family and the Spartan 3ADSP family that is oriented to DSP applications and includes DSP slices. A peculiarity of Spartan 3 FPGAs is that, they do not include on-chip tri-state buffers, as does most of the high performance Virtex counterpart.

Like other vendors, Xilinx provides two approaches for embedded processing: hard core where the processor is embedded in the FPGA die and soft core processors where the processor is implemented with FPGA logic. As hard core processors, there are two versions of the IPB power PC 32-bit RISC processor that are being used: the PowerPC PPC405, currently supported by the Virtex II Pro, and the PPC440 embedded in the Virtex 5 TXT FPGAs. As soft core processors, Xilinx provides MicroBlaze and PicoBlaze. MicroBlaze is a flexible 32-bit RISC processor that can be implemented on any Xilinx FPGA, from the low-cost Spartan 3 to the high performance Virtex5 FPGA. The PicoBlaze microcontroller is a compact 8-bit RISC microcontroller core optimized for the Spartan 3, VirtexII, and VirtexII Pro.

In FPGAs all user-programmable features are controlled by memory cells that are volatile and must be configured on power up. These memory cells are known as the configuration memory, and define the LUTs equations, signal routing, IOBs, voltage standards and all other aspects of the design. The configuration memory can be visualized as a rectangular array of bits. The bits are grouped into frames that are the atomic unit of configuration - a frame is the smallest portion of the configuration memory that can be written to or read from the configuration memory.

A distinguishing characteristic of Xilinx FPGAs that has made them a de-facto choice for dynamically reconfigurable systems is their reconfigurability features: they support dynamic partial and self-reconfiguration. For making possible self-reconfiguration, Xilinx provides, on the FPGA die, an Internal Configuration Access Port (ICAP) so the proper FPGA can control the reconfiguration process. Furthermore, with each new FPGA generation, Xilinx is improving its reconfiguration features, now Virtex 5 FPGAs have enhanced partial reconfiguration capabilities.

To program the configuration memory, instructions for the configuration control logic and data for the configuration memory are provided in the form of a bitstream, which is delivered to the device through the JTAG, SelectMAP or ICAP configuration interface.

1.3.1.2 Altera

Altera, like its main competitor, Xilinx, offers several FPGA series, high-end Stratix devices and low cost Cyclone. The Stratix series of FPGA families is composed of several device families, with different features briefly described below:

1. Stratix and the Stratix GX families are the first members of the Stratix FPGA series (130-nm technology). According to Altera, these are the devices that have introduced the use of DSP hard IP blocks.
2. Stratix II and Stratix II GX FPGA families have introduced the adaptive logic module (ALM) architecture, which uses high-performance LUTs. This, second, Stratix generation includes a variation (GX) that has Gbps transceivers and has lower power compared with the predecessor one, as they are designed on 90 nm technology.
3. Stratix III FPGAs are the Altera low power high-performance 65-nm FPGAs. This family has three branches, one with enhanced memory (L) series, another has extended memory (E) and the last one has digital signal processing (DSP) blocks. These FPGAs target high-end core system processing designs in many applications.
4. Stratix IV FPGAs are the fourth generation Stratix FPGA family and provide high density, high performance and low power in a 40-nm technology.

Apart from this, Altera provides a 32-bits RISC general purpose soft core processor, Nios II. They also provide a few years ago a solution for embedded processing based on a 32 bit ARM microcontroller that has been included in the silicon die of the APEX 20KE family.

1.3.1.3 Atmel

Atmel offers two SDRAM based FPGA families, the AT6000 and the AT40 [Atm]. The Atmel's AT40KAL families of Field Programmable System Level Integrated Circuits (FPSLIC devices) combine basic system building blocks like logic, memory and microcontroller in an SRAM-based monolithic field programmable device. In addition to the FPGA these devices have an on-chip serial configuration memory. The combination of FPSLIC and the serial configuration memory in a single package allows in-system programmability (ISP) and remote device updates by modifying the serial configuration memory. FPSLIC devices combine 5K to 50K gates of Atmel's patented AT40K FPGA architecture, 20 MIPS 8-bit RISC microprocessor core, fixed microcontroller peripherals and up to 36K Bytes of program and data SRAM. The AT6000 series FPGAs are designed to speed up processor-based system performance while, lowering power, part count and cost. The massive register counts (1,024 to 6,400 registers) make them ideal for use as re-configurable DSP co-processors.

1.3.1.4 Lattice

Lattice semiconductors owns the world's first non-volatile, infinitely reconfigurable FPGA called ispXPGA [Lat]. This device contains two types of memory, Static RAM and non-volatile EECMOS cells. The static RAM is used to control the functionality of the device and the EECMOS cells are used to load the SRAM memory. This allows a dynamic full reconfiguration of the device by reconfiguring the EECMOS memory cells, while the rest of the device is operating. Partial reconfiguration, however, is not possible. There are two possible ports that can be used for configuration of the SRAM memory; the JTAG (IEEE 1149.1), Test Access Port (TAP) - bit-wide configuration and the Peripheral port (parallel). On the other hand, when programming the E2CMOS memory, only JTAG can be used. Lattice also provides an open-source 32-bit Harvard, RISC architecture soft core microprocessor.

Another worthy to mention FPGA provider is MathStar [Mat]. This company has proposed a new architecture paradigm based on medium-grained processing elements, objects, called Field Programmable Object Array (FPOA) that supports clock frequencies of up to 1 GHz.

1.3.2 Custom Reconfigurable Devices and Platforms

Basically, reconfigurable platforms provide high performance while keeping, theoretically, software flexibility. General purposes, fine grain reconfigurable devices, like the FPGA described in the previous section, provide these flexibility features to any application.

Differently, if the target application or application domain is known in advance, a more suitable reconfigurable system with higher performance, but with flexibility penalties can be identified.

Several custom devices and custom platforms, at different granularity levels and with different coupling, have been proposed. Generally, these custom platforms target a narrower set of applications achieving reduced power and improved performance. The list of custom reconfigurable devices, systems and platforms is quite extensive. Here, in the next paragraphs, some representative solutions from the academic community are described.

In Multi FPGA systems, the basic building units are commercial or proprietary fine grain FPGAs. Several FPGAs are connected in large boards resulting in standalone high performance platforms connected to host microprocessor systems [Miy98]. One of the first reconfigurable platforms that have reported significant performance achievements in the early nineties were the Splash [ABD92] and the PeRLe-1 [bJVBR⁺96].

The Splash 2 system, an enhanced version of Splash, contained 17 Xilinx XC4010 FPGAs, 512 Kbytes memory and crossbar switches used to connect two processing elements. The system higher performance (10,000 times faster than a workstation) was achieved with a DNA pattern search applications [Arn93]. The PeRLe-1 [bJVBR⁺96] platform, composed of 24 XC3090 Xilinx FPGAs, has been successfully applied mainly to solve problems related to encryption, video compression, high-energy physics and sound synthesis.

Other fine grain solutions rely on the design of new device architectures, like the PipeRench chip from the Carnegie Mellon University (USA), Chimaera [HFHK04] or DPGA [DeH94]. The PipeRench device, for instance, is one of the first to introduce the virtualization of hardware in order to enhance the re-usability of applications. The last property is accomplished by providing the device with self-reconfigurable features [HDA⁺02]. The target application domain for this device was numerically intensive applications.

An interesting system is Teramac. It is an experimental parallel computer designed by HP in the 1990s [ACC⁺95]. The basic idea was to provide a programmable system with million gates, running at a megahertz, and based on defective processors and FPGAs. The FPGAs used were several custom PLASMA chips. PLASMA is a routing-rich FPGA that consists of 6-input, 2-output LUTs with configurable registers, interconnected by crossbar switches. Although the system had 220,000 hardware defects, speedups of 100 times compared to single-processor high-end workstation were reported. These parallel, multi FPGA, systems are suitable for deeply pipelined and highly parallel bit-oriented computations. Additionally, their basic fine grain solutions provide an enhanced flexibility when implementing other tasks.

In the coarser grain range, stand alone solutions are oriented to specific application domains, among others: data processing, DSP application and cryptography, like the Chameleon cipher [SKW⁺07]. Further solutions, also in the coarse grain range, use a reconfigurable coprocessor or a reconfigurable functional unit [CBN08]. In this case, the reconfigurable hardware is tightly coupled to the microprocessor, like in Morphosys from the University of California, Irvine, MOLEN from TU Delft and ADRES from IMEC. In these approaches the reconfigurable array configuration is generated after an application code analysis. The application source code is profiled and the computational heavy parts are ported to the reconfigurable array. The coprocessor configuration is generated at design time, along with the software program. This permits to accelerate a specific application achieving high performance and reducing system power consumption. MorphoSys, for instance, targets computing intensive and parallel data applications [SLL⁺00]. The architecture is composed of a software programmable processing unit called TinyRISC and an 8x8 reconfigurable hardware unit called RC Array. Each RC has a 16-bit datapath and contains a multiplier, an ALU, a shifter and a small register. RCs are configured by context words, which specify an instruction core for each RC. As MorphoSys targets multimedia application, cells on the same row receive the same control word, permitting instructions over 128-bit words. ADRES stands for Architecture for Dynamically Reconfigurable Embedded Systems [BBSG08], [BBKG07]. The system has been developed in the context of IMEC's multi-mode multimedia program and targets data processing in mobile terminals. The ADRES architecture couples a VLIW processor with a custom coarse-grain-array accelerator, through a shared central register-file. Afterwards a specific ADRES instance is generated for a given application, like MPEG-2, MPEG-4 and H.264/AVC. Differently, the currently available MOLEN solution is based on commercial Xilinx Virtex II Pro FPGAs [VWG⁺04] and [PBV07]. Computational heavy tasks are enclosed in a microprocessor peripheral connected to the system bus. This coprocessor is implemented with the FPGA fine grain array and special instructions are included in the on-chip embedded microprocessor to deal with this hardware.

1.4 Partial Runtime Reconfiguration

The constant pressure on the electronics market makes the use of FPGAs advantageous due to a reduction of the product time-to-market and the possibility of system hardware updates after deployment, as it has been explained in section 1.1. Furthermore, today's advanced FPGAs are complex reconfigurable platforms, composed of a broad set of reconfigurable elements that can be reprogrammed using several methods, as it has been described in section 1.2. New technologies, like the possibility of partially reconfigure a device takes the powerful benefits of reconfigurability to a much higher level and permits to overcome restrictions related to long configuration times and high power consumption when reconfiguring the entire device.

The general benefits of using partial runtime reconfigurable systems [NDG05], [Kao05] and [GDS06], instead of conventional reconfigurable devices are briefly summarized next:

1. *Increased system performance.* The system remains fully operative, while a portion is being reconfigured. Consequently, as the system does not need to be stopped and theoretically, the data flow through the FPGA can continue, performance is improved.
2. *Enhanced system updates.* By modifying only a portion of the device, system updates do not affect the remaining devices connected to the FPGA. They can keep functioning normally, as long as the IOBs are not affected by the reconfiguration.
3. *Hardware sharing.* Partial reconfiguration allows not only having several applications sharing the same FPGA, permitting to select smaller and cheaper devices, but also to run them in parallel (concurrently). This characteristic is gaining importance with the increasing FPGA integration level.
4. *Shorter reconfiguration times.* Partial reconfiguration deals with smaller configuration times compared to full device updates. This results in reduced configuration times and also in reduced power consumption during reconfiguration.
5. *Less storage resources.* Partial configuration files require less storage resources and this is important for resource restricted devices.

Partial reconfiguration is valuable for devices that operate in environments where applications cannot be disrupted while the system is being adapted. And also, for highly parallel systems that can time-share the same piece of the FPGA resources. Without this capability, it would be necessary to stop the system during device updates and to reconfigure the entire FPGA to support a different application, losing all other (previous and current) configurations.

From the research point of view, partial reconfiguration is very tentative as this technique opens the way to create highly flexible self-reconfigurable, self-arrangeable, evolvable, cognitive and/or cooperative hardware systems, among others. All this sounds very promising, but partially reconfigurable systems suffer because of the lack of flexible solutions and user friendly design flows and tools. Furthermore the reduced

variety of commercial partially reconfigurable devices and FPGA vendors' poor support make the design possibilities quite restricted. The thesis work, presented in this document, attempts to contribute to improvements in the flexibility of the systems by providing methods and tools.

1.5 Partially Reconfiguration Systems Based on Commercial FPGAs: Prospective

This section covers with more detail and discusses some of the current restrictions when designing partially reconfigurable systems based on commercial FPGAs.

1.5.1 Dynamic Reconfigurable Commercial Devices: Selection

Before starting the design of partial runtime reconfigurable systems on FPGAs, it is necessary to select a device that supports this reconfiguration technique.

Altera has the second position in the FPGA market. Altera Excalibur devices allow dynamic configuration of the FPGA while the on-chip processor is active for other tasks, such as running the operating system. A user can alter the device functionality by storing the configurable system function in a nonvolatile memory and use the processor to configure the hardware without the need to reboot. This results in systems with a single reconfigurable coprocessor that is composed of the entire reconfigurable array. Additionally, more recently, some Altera FPGAs permit to reconfigure specific elements, like SerDes or the PLLs. However *partial dynamic reconfiguration is not supported*.

Differently, Atmel FPGAs have the ability to implement Cache Logic designs, where part of the FPGA can be reprogrammed without loss of register data, while the remaining of the FPGA continues to operate without disruption, that is real partial reconfiguration. Anyway, the main drawbacks of these FPGAs are: i) their size, which is quite small (5K to 40K gates) compared to other and ii) the method used to access the configuration resources that is bit based [ATM05] and requires very low level control of the reconfiguration process.

Last but not least, Xilinx FPGAs can be partially reconfigured. Two factors made possible the implementation of dynamic partial reconfiguration in the Xilinx FPGA structures: the packet structure of the configuration bitstreams that allows a designer to modify one or more configuration packets and perform partial reconfiguration, and the internal structure that permits direct access to the configuration memory of the FPGAs. Apart from this, each device family has different, added, features that are discussed next. The Spartan 3 series permit to reconfigure entire device columns that span from the FPGA top to the bottom side of the FPGA floorplanning. Therefore, when a single device row has to be changed then, the information for the entire column has to be sent to the device program memory. Another, specific, characteristic is that glitches can appear during reconfiguration in the device circuit, even if no changes are being made in the program memory (the loaded configuration is the same). Furthermore, the first Spartan 3 family does not include an ICAP and thus it is not the best suited for

designing self-reconfigurable systems. Differently, the subsequent Spartan 3 variations, Spartan 3A, Spartan 3AN, Spartan 3ADSP include such port.

On the contrary, all families in the high performance FPGA series permit glitch-less reconfiguration and include one or two ICAP configuration port. This makes these FPGAs well suited for designing partial run-time reconfigurable systems. However, the different device families provide different partial reconfiguration techniques. In the Virtex II and the Virtex II Pro families, like in Spartan 3 FPGAs, a configuration frame spans the entire FPGA height. Even though, they are the most extended families for runtime reconfigurable system design nowadays. The latest FPGAs, Virtex 4 and Virtex 5 families, have some important improvements. In these FPGAs, a frame does not span the entire FPGA height, but several FPGA rows/columns (16x16 in Virtex 4 and 20x20 in Virtex 5). Furthermore, Virtex 5 FPGAs include two, faster, ICAPs. These improvements are probably a result of the research community efforts in designing architectures, tools and applications based on the partial reconfiguration technique and demonstrate the technique advantages. Anyway, the mentioned device improvements, as well as the given support, are not significant enough and thus, the migration to the new platforms is being relatively slow. Nevertheless, the newly starting works are based on such devices.

From the exposed data, it is clear that supporting partial reconfiguration in today FPGAs is not a common aspect. Currently, both Atmel and Xilinx provide true partial dynamic reconfiguration. Anyway, due to limited capacity of Atmel's FPGAs and the enhanced programming features of the Xilinx ones, *the selection of Xilinx FPGAs for designing partial runtime reconfigurable system based on commercial FPGAs is forthcoming. More than 90 % of the work in this area, including this thesis, is based on them.* Furthermore, at the time this thesis work started, the state of the art Xilinx devices were Virtex II and Virtex II Pro and therefore these FPGAs will be targeted along the thesis.

It is highly important to mention some relevant works, based on ATMEL's FPGAs, like [BBP04], [KBD07] and [LMGVE04], [LMME07]. The last one, for instance, presents a reconfigurable system based on an ATMEL AT40K FPGA and demonstrates power reduction compared to no partial reconfiguration solution targeting data processing systems.

1.5.2 Hardware Abstraction

Hardware abstraction or virtualization has been used in some cases to overcome problems related to reconfigurable systems deficiencies, like long reconfiguration time and poor system portability.

Increasing system portability is targeted in [DMC⁺06], where a virtual hardware is defined following a software paging approach. In this work, a reconfigurable page is a piece of the FPGA (called paged array) that also includes the IOBs and does not address problems related to on-chip interconnections. Furthermore, in [Sek07], hardware virtualization is related to mapping a reconfigurable circuit on top of the FPGA (virtual circuit) in order to reduce high reconfiguration times. It is interesting to mention that realtime reconfiguration rates have been achieved by reducing the delay

to one clock cycle. However, this abstraction technique has high area penalties.

In contrast, this work is oriented to the virtualization of the reconfigurable device architecture. This virtualization, differently to other virtual hardware concepts, targets the system logic distribution and interconnections. This virtualization, called Virtual Architecture (VA), defines the reconfigurable area boundaries, and how they are interconnected and leads to the design of scalable and flexible reconfigurable systems and permits hard core reuse as it will be demonstrated in this document.

From here after, each portion of the FPGA that can be partially reconfigured in the context of this thesis will be referred as a slot.

1.5.3 On-chip Communications

There are three types of interconnections that can be defined among slots: direct (point to point-P2P), Bus and NoC types. Each type is briefly described below:

1. Direct point to point connection. This connection technique is widely used when a small amount of reconfigurable slot(s) are defined in the system [CL07]. This is the unique on-chip communication solution that is covered by FPGA providers. Therefore, the flexibility and scalability of reconfigurable systems that can be designed following the commercial solution, as it will be further explained in the following thesis Chapters, are restricted.
2. Bus communication. This connection technique is used when a single or several slots, that act as bus masters, control data transactions with slave slots. Related to this, a direct approach in tightly coupled reconfigurable systems is to attach slots to the microprocessor bus. Some academic examples of such systems are [QTSN08], [OMFK08] [CZMS07] and [FS05]. In the last work, for instance, several reconfigurable modules are clustered and then, attached to the microprocessor bus which acts as master. Other approaches define a proprietary on-chip bus, like [MHB05].
3. Networks on Chip (NoCs). In the last years, NoCs have appeared to solve problems related to bus communication. NoCs are a paradigm shift where cores are interconnected through short, point to point, links and data is transmitted in packets. This permits to reduce the topology-sensitive global interconnect delays that do not scale with device switching times [ITR03]. On the contrary, studies have shown that NoCs are beneficial in terms of area-performance tradeoff only when the number of interconnected cores is high [WG02]. As a reconfigurable system on-chip interconnection option, NoCs for instance have been selected to interconnect system cores in [MNM⁺04], [JBH], [HCG07], [BA05], [TPB07] and [OMFK08].

On-chip communications will be discussed along the thesis and it will be explained (in Chapter 3) that flexibility at the communication level is needed for reconfigurable systems. One of the aims of this thesis is to explore reconfiguration capabilities not only in the logic related to cores, but also on the communications, and so, an architectural solution based on NoCs will be originally proposed in Chapter 3.

1.5.4 Design Methods and Software Support

Apart from the device selection and the reconfigurable system architecture definition, very important issues are: the software development tools and the reconfiguration control. Software development tools and flows are needed to design hard cores for partial runtime reconfigurable systems. The commercially provided solutions, like the Xilinx modular design [Eto07], [DL04], and the Early Access Partial Reconfiguration Flow [Xil06a], [Xil06b] cannot cover reconfigurable systems designers requirements. The main drawbacks of this tools and flows are:

1. The general and most important drawback is that systems designed following the Xilinx methods and tools, are quite restricted and require a firm relation between the system architecture and hard cores to be loaded. For instance, if a reconfigurable device has to be updated after deployment, by loading a hard core, the hard core has to be specifically designed for that particular reconfigurable system and also, for the place it will occupy on the FPGA.
2. Another important disadvantage is that the provided tools are not flexible in terms reconfigurable systems runtime support. The available tools do not permit any kind of hard core manipulation, being the most important the possibility of allocating a hard core in a different FPGA position, which will be referred from here after as core relocation. As a result, reconfigurable devices cannot autonomously control its configurations.
3. The Xilinx flow (or part of it) has to be repeated several times to generate all configurations of the reconfigurable systems (the full initial configuration and all the required partial configuration files).
4. Designers require specialized knowledge of partial reconfiguration.
5. An integrated solution is not provided and thus several tools have to be used along the flow.
6. Currently provided flows do not solve problems related to reconfigurable systems simulation and debug.
7. Other highly important disadvantage of the commercially available FPGA design tools is that the provided place and route tools permit to define only placing, but not routing, restrictions. Therefore it is not possible to restrict design routing in a defined bounding box, resulting in wires crossing the defined box borders. This restriction will be referred in the document as boundary crossing and it will be shown that it has an important impact on reconfigurable systems. As this restriction is related to design tools specifics, and not to the design method, it will not be addressed in this thesis.

Consequently, many research groups have defined their own design methods, flows and tools, subject to their specific needs and focused to solve one or several of the listed problems.

For example, in [SS06] and [DFR⁺05], the "Caronte" design flow is presented, which is based on the Xilinx solutions and provides architecture support for Virtex II and Virtex II Pro devices. In the "Caronte" design flow, the idea is to consider the system in time as a sequence of static photos where all the possible HW configurations share the static part of the system. Other solutions, also based on the Xilinx approach, are the Proteus framework [WN04], the RECONFIG2 project framework [BR04] and, the PaDReH framework [CCBM04] that starts the reconfigurable system design at a higher abstraction level (System C). These, Xilinx based solutions, solve problems related to providing integrated and friendly flows that start at different abstraction levels, and in some cases also include simulation features. Differently, non Xilinx based solutions, that target to provide enhanced system flexibility and permit to manipulate the FPGA configuration file in order to adequate a hard core to be loaded in a system are for instance, in [PHP⁺04], [RS02], [SF06a] and [HL04].

This thesis work belong to the second group of solutions, not based on the Xilinx approach. *The main and general approach to be followed in this thesis work is to divide the system design process from the hard core design, in order improve system and hard core design flexibility and scalability. To achieve this goal, the thesis work addresses issues related to system virtual architectures design, bitstream manipulation and also, problems related to the flow repetitiveness and the required designers' expertise.* Problems related to reconfigurable systems debug are also targeted, but through an example application in section 5.4, and are not part of the main topic.

With respect to the runtime control, the commercially available solutions barely cover loading and reading back configuration files from the FPGA. As a result, a deployed system can consume only hard cores that are specially designed for it and therefore, device independence is not straightforward. Processes related to reconfiguration control systems and more specifically to bitstream manipulation issues and reconfigurable system abstraction, are not solved by FPGA providers. Therefore, several academic solutions have been adopted: from simple control systems, implemented either in HW [KP06] or in a SW running in the embedded processor (like one of the tools proposed in Chapter 3), to complete hardware operating systems based solutions, like [WP04]. Furthermore, some solutions extend an already existing operating system, like Linux, [NMV⁺04], [KMK07], [BT04].

In this thesis, a solution for hard core proper allocation in embedded devices, that targets Virtex II FPGAs, will be originally proposed in Chapter 3. The runtime control issue will be tackled from the practice. In two of the four applications included in Chapter 5, a specific reconfiguration control will be defined. For the less flexible system, the runtime control will be simply a software tool in charge of loading new configuration files in the FPGA. Differently for the second application, a complete reconfiguration control middleware will be integrated in a Linux operating system. In this case, the reconfiguration control is oriented to the integration of the reconfigurable device in a client-server environment where the system can consume different hard cores. For the remaining applications, no specific runtime control is foreseen.

A complete reconfigurable system solution is the one that is composed of all the previously described elements: architecture definition, hard core development tools and reconfiguration control. Example of such solutions are [dM], [JWLZ03], [MTAB07], [SF06a] and [MNM⁺04]. *In this thesis, several reconfigurable systems, based on the*

virtual architecture design method originally proposed in Chapter 2 and the support tools presented in Chapter 3, have been arranged and will be originally presented along the applications section in Chapter 5.

1.6 Summary and thesis Objectives

Designing a complete reconfigurable system nowadays is not an easy task. The commercially available solutions are quite restrictive, complex to use and require specialized designers. This complexity starts from the basics of the reconfigurable systems design methods, the hardware infrastructure, goes through hard core design flows, and ends by the runtime reconfiguration control. New solutions in all these aspects, oriented to bring flexibility and scalability to the system and to provide end to end reconfigurability are required. *The main approach to be followed in this thesis work is to divide the system design process from the hard core design, in order to design highly flexible and scalable reconfigurable systems. The general goal is to permit a reconfigurable device to consume a board set of hard cores on one side, and on the other side, to permit the design of hard cores without the need of knowing the reconfigurable system specifics.* To achieve this, new solutions, related to system design method and tools, as well as system runtime control, will be provided along the thesis and it will be shown that the provided solutions lead to the design of more flexible and scalable reconfigurable systems compared to the state of the art.

The main, detailed, objectives of this thesis work are listed below:

1. *To provide a solution to the design of virtual architectures for reconfigurable systems* that might permit to: i) design one and two dimensional flexible systems that permit hard cores to be freely allocated along the defined architecture, ii) improve the use of the FPGA resources and iii) permit stable reconfiguration (the on-chip communications have to remain active during the hard core hot swapping process). These goals are the main subject of Chapter 2.
2. *To extend the system reconfigurability and provide reconfiguration not only of the system cores, but also to the on-chip communications.* In Chapter 3 a NoC-based architecture solution that covers this goal will be originally proposed, along with a set of reconfiguration methods and suitable design resources that include, among others, an original packet format and a router architecture approach.
3. *To define a set of tools and design flows* that will provide support to the system flexibility. On one hand, the main objective for the design tools is to target embedded systems in order to provide higher independence and permit the system to consume different hard cores. On the other hand, for the design flows, the main objective is to facilitate the reconfigurable system design process, reduce the need of expert designer and detailed knowledge of the target reconfigurable system architecture, while keeping the provided solutions as close as possible to existing commercial tools. This topic is targeted in Chapter 4.
4. *To demonstrate the feasibility and flexibility of the provided solutions, several systems with a different flexibility level have been created and integrated in four*

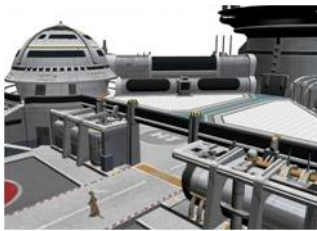
different application domains: i) to a remote reconfigurable sensor network node, ii) to an automatic and transparent remote reconfigurable client device integrated in a client-server environment, iii) to debug a reconfigurable system and iv) in a complete on-chip communication emulation framework. The diversity of the selected application domains, described in Chapter 5, is a result of the search of possible “killer application” domains of partial runtime reconfigurable systems.

At this point it is highly important to remark that the amount of groups working in the different subjects of the reconfigurable systems research topic is constantly increasing. Nowadays, the competitiveness and the advances in this topic are impressive. While a few years ago, the research groups in this topic were mainly in Europe and in USA and it was relatively easy to track all the progresses, now there are a lot of contributions from all over the world, being very active Asia. Consequently, the amount of published work is much larger than the included along the thesis.

For being more precise, and due to the board of topics to be discussed along the thesis, there is a state of the art (introduction) and a results part in each Chapter (but the applications one). Along the introduction section, a few selected works will be described in detail and compared with the proposed solutions in the Chapters’ results section. *The selected works for each Chapter’s state of the art have been, in most of the cases, the contemporary ones and the closest related to this thesis.*

Chapter 2

Reconfigurable Systems for Commercial FPGAs



Architectures of partially reconfigurable systems. A method for designing Virtual Architectures for partial RunTime Reconfigurable Systems (pRTRSs) is originally proposed.

The basics of the flexibility and scalability of a reconfigurable system are defined by its virtual architecture, which defines how the FPGA resources are distributed in order to map the systems, and how the different regions are interconnected. This Chapter *proposes a general method for Virtual Architectures (VAs) design, which is a thesis original contribution.*

In the first part of the Chapter, section 2.2 and section 2.3, existing partially reconfigurable systems design solutions from the industry and the academic community are presented.

In the second part, section 2.4, the virtual architectures design method is extensively described and used to build several, general, reconfigurable systems for commercial FPGAs. Further, two one dimensional bus based reconfigurable systems are presented and compared with the state of the art at the end of the Chapter in the results section 2.5, outlying the advantages of the proposed method. Finally, conclusions can be found in section 2.6.

2.1 Introduction

A time diagram of the research groups that will be included in this Chapter state of the art described in the next subsections can be found in Figure 2.1. The works included in the Figure and in this section are related to one dimensional (1D) runtime reconfigurable systems. It is worthy to highlight that the main part of the presented in this Chapter related work are focused on bus based one dimensional systems and has been published between 2003-2006. After that, a tendency to move to Networks on Chip (NoCs) and two dimensional (2D) reconfigurable systems can be noticed, as it will be explained in Chapter 3, where a similar time diagram will be presented and the state of the art part is focused on 2D systems.

Although, the method proposed along this Chapter is valid for both possible models (1D and 2D), for clearness this Chapter will extensively present the method along with a general application to commercial Xilinx FPGAs and a design of two 1D bus based reconfigurable systems that will be compared at the end of the Chapter with other 1D systems from the state of the art.

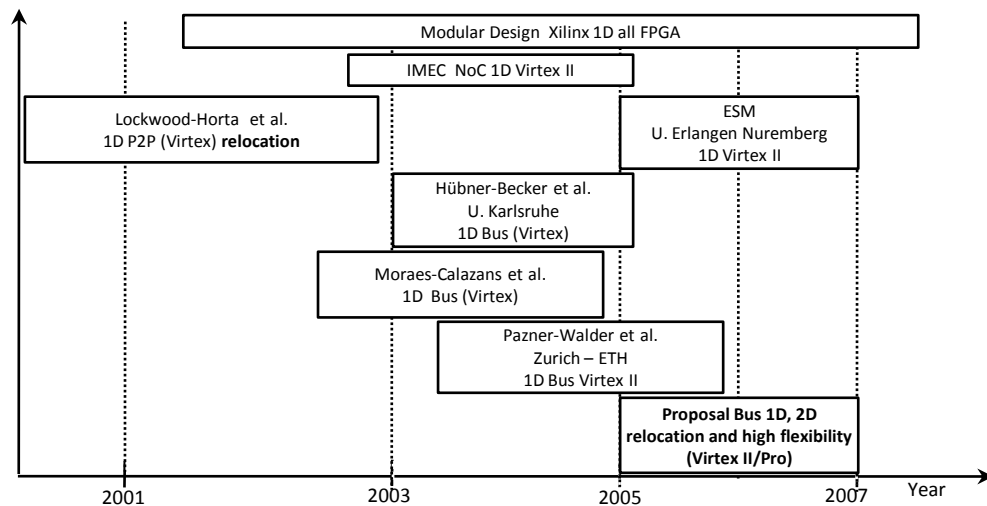


Figure 2.1: One dimensional partially reconfigurable systems evolution time diagram.

2.2 Commercial Approach

As it has been explained in the introduction part of this thesis, nowadays the most feasible solution for building reconfigurable systems with commercial FPGAs is the one provided by Xilinx. Therefore this section is focused on the official guidelines for reconfigurable systems design, provided by Xilinx, published in [DL04] and valid for all its FPGAs.

2.2.1 FPGA Resource Division

The first step that has to be done for building a reconfigurable system is to define the area of the FPGA that will be reconfigurable. Therefore, the FPGA is divided in fixed and reconfigurable areas. In the Xilinx approach the fixed area spans the two leftmost and rightmost FPGA CLB columns of the FPGA floorplanning and the rest of the FPGA, the reconfigurable area, allocates the reconfigurable modules. According to [DL04], each Reconfigurable Module (RM) has the characteristics listed below:

1. The reconfigurable module height is always the full height of the device.
2. The reconfigurable module width ranges from a minimum of four slices to a maximum of the full-device width, in four-slice increments (modules width of 4, 8, 12, .. slices).
3. The horizontal placement of reconfigurable modules must always be on a four-slice boundary; the leftmost placement being $x = 0, 4, 8, \dots$
4. All logic resources encompassed by the width of the module are considered part of it. This includes slices, TBUFs, block RAMs, multipliers (MUL), IOBs, and all routing resources.
5. IOBs immediately above the top edge and below the bottom edge of a reconfigurable module are part of the specific reconfigurable module resources, but can be used as module I/Os.
6. The number of reconfigurable modules should be minimized (ideally, just a single reconfigurable module).
7. Reconfigurable module borders and position cannot be modified once defined.

An example of a reconfigurable system, built following the previously listed reconfigurable modules characteristics and composed of two reconfigurable modules, is shown in Figure 2.2.

With the described approach, simple reconfigurable systems based on one dimensional (1D) resource distribution models can be built. These systems have poor flexibility regarding modules allocation/relocation and poor scalability, regarding the amount of reconfigurable modules that can be defined and their interconnections (only neighboring links are foreseen). As a result, reconfigurable systems, designed with this approach require the use of not only architecture, but target reconfigurable module,

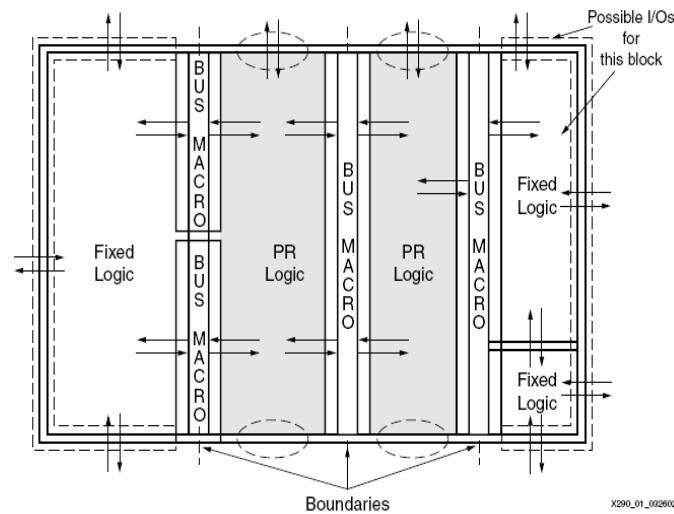


Figure 2.2: Example reconfigurable system with two reconfigurable modules, defined using the Xilinx recommendations [DL04].

specific hard cores. Furthermore, hard core reusability along different reconfigurable systems is impossible.

2.2.2 Communication Structures

Partial reconfiguration requires that signals used as communication wires between or through reconfigurable modules use fixed routing resources that must not change when a module is reconfigured. According to [DL04], this is achieved by using special structures, called Bus Macros (BM), that guarantee fixed wires for signals going between a reconfigurable module and another module (reconfigurable or non reconfigurable).

At design time, the HDL code should ensure that any reconfigurable module signal communicate with another module only passing through a BM. This types of macros, presented by Xilinx in [DL04], will be referred as BM-TBUF-Xilinx. An example Figure of a reconfigurable system with highlighted bus macros position can be seen on Figure 2.3.

The implementation of Xilinx BM-TBUF-Xilinx uses eight tri-state buffers (TBUFs), four from either module involved in the communication, hooked up in an arrangement that allows four bits of information to travel either left-to-right or right-to-left (see right side of Figure 2.3). For Virtex II devices, each BM-TBUF-Xilinx occupies two CLB columns from each communication side (there are two TBUFs available in each CLB). As many bus macros as needed must be instantiated to match the number of bits traversing the boundaries of the reconfigurable modules. For instance, if reconfigurable module A communicates via 32 bits to its neighboring module B (see Figure 2.3 - left side), then eight ($32/4$) BM-TBUF-Xilinx will need to be instantiated to define the data path between both modules.

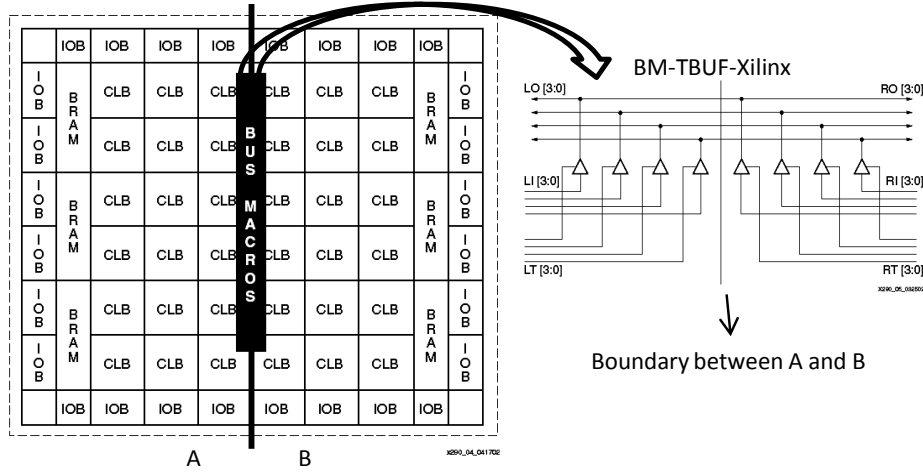


Figure 2.3: Xilinx reconfigurable system on-chip communication structure - BM-TBUF-Xilinx. Communication structure position on the left side and a detailed view on the right side.

The main drawbacks of using BM-TBUF-Xilinx that have been noticed after some test implementations are that: first, TBUFs horizontal routing lines span four CLB columns and therefore another BM-TBUF-Xilinx could be placed in the same FPGA row only if it stands at a minimum distance of two CLB columns. Second, BM-TBUF-Xilinx signal direction is defined at design time and, if all the four signals are selected in one direction, due to switch matrix routing restrictions, four CLB columns will be needed in the receiver side resulting in reconfigurable modules width restriction. In the same document, [DL04], a complete design flow for partial bitstream (pBS) files generation, called Modular Design is included. This design flow will be deeply discussed in the related work section of Chapter 4.

It is worthy to remark that more recently, partial reconfiguration flexibility has been enhanced in [Xil06b] and [Eto07], where different types of CLB based macros, some of them similar to the ones that will be described later in this Chapter, have been introduced. Also, the provided support software, the Plan Ahead tool, differently from the modular design [DL04] permits to define square reconfigurable modules if they are surrounded by fixed regions. Furthermore, wires that belong to the fixed area can cross reconfigurable modules without the use of Bus Macros. This reduces former routing restrictions, but relocation is prohibited and the scalability problem is not solved. Solutions for flexible reconfigurable module allocation and relocation, as well as architecture scalability have not been provided so far by Xilinx.

Finally, new Xilinx FPGAs, Virtex 4 and Virtex 5, permit block reconfiguration where blocks of 16x16 and 20x20 CLBs, respectively, can be independently reconfigured. In such FPGAs, a two dimensional model (2D) could be directly mapped with the restriction that a reconfigurable modules have to span all the 16x16 or 20x20 CLBs resulting in large reconfigurable modules of 400 CLBs. However, although the extended partial reconfiguration features, the provided flows and tools in [Eto07] and [Xil06b] do not fully exploit this new capabilities.

2.3 Academic Approaches

In this subsection, some relevant academic works in the area of partial runtime reconfigurable systems will be reviewed.

2.3.1 Lockwood-Horta et al.

The Washington University, in cooperation with Xilinx, has created a platform for reconfigurable systems design oriented to telecommunications and called Field Programmable Port eXtender (FPX) [JWLT00].

The target applications of this platform are: i) rapid prototyping of telecommunication routers in [FBL01] and ii) firewalls in [JWLZ03]. The FPX platform is composed of several FPGAs and one of them, a Xilinx XCV2000E (Virtex E), is the **Reconfigurable Application Device - RAD**. The reconfigurable area of this FPGA holds reconfigurable modules called, Dynamic Hardware Plugins (DHPs) [ELH02]. A schematic view of the RAD FPGA with two DHPs can be seen on the left side of Figure 2.4. DHPs are used to implement application-specific functionality on the FPX platform. The system permits multiple DHPs to be loaded into the RAD and run in parallel on a single FPGA device.

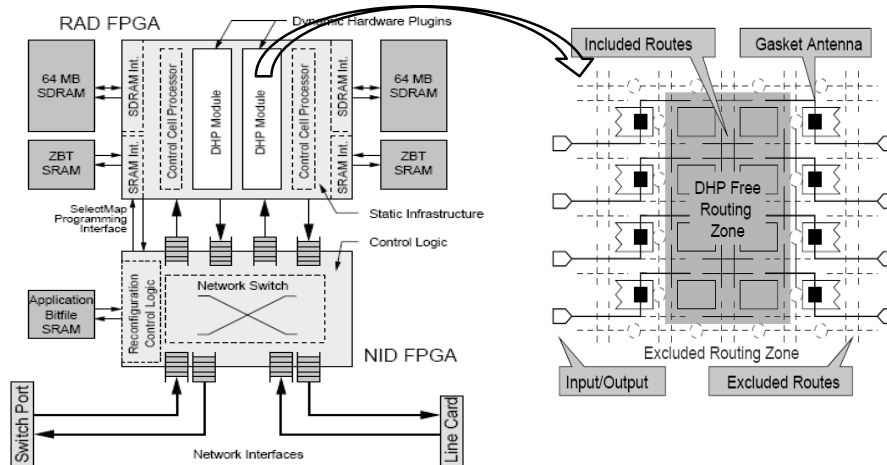


Figure 2.4: Lockwood-Horta et al. On the left side, the Reconfigurable Application Device (RAD) with two reconfigurable modules called Dynamic Hardware Plugins (DHP). On the right side, a detailed view of the DHP module fixed interfacing signals called "antennas" [ELH02].

The DHP interface guarantees the DHP communication with: i) an on-board SDRAM and DRAM memories, ii) an Input Output Controller (IOC) used for communication with other DHPs and iii) a central application controller [TTLH02]. The IOC, the application and memory controllers are distributed in the left and right FPGA sides and are part of the FPGA fixed area as it can be noticed from the floorplanning shown on the left side of Figure 2.4 where it is marked as "static infrastructure".

The Lockwood-Horta et al. solution for the problem of guaranteeing fixed interconnections (explained in the previous subsection and solved by Xilinx with bus macros) is called "Gasket Interface". Gasket interfaces define fixed connection points that are linked by special wires called "Antennas". Differently from other solutions, the Lockwood-Horta et al. antennas are not based on pre-routed communication macros. Antennas are created by a set of HDL signal assignments and D Flip-Flops. The antennas placement and routing is guaranteed by defining placement restrictions and reserving an area around the DHPs that is prohibited for placing logic during the place and route process of the design flow. Such area can be seen on the right side of Figure 2.4 marked as "excluded routes" zone.

In [ELH02], a demonstration of the RAD system with two independent DHPs is presented. There, each DHP is independently connected to on-board LEDs. One DHP to LEDs on the board right side and another on the left side. The presented application uses partial reconfiguration of the DHPs to modify the LEDs ON/OFF sequence.

In an overall evaluation of the Lockwood-Horta et al. 1D architecture, it is noteworthy that the amount of connection between DHPs and the fixed areas is very high (more than 200). This limits the amount of DHPs that can be defined (two DHPs are reported) and restricts DHPs relocation along the reconfigurable area.

Finally, it is important to remark that from the latest publications of this group it can be noticed that this research line is active. In [EAGEG07], partial reconfiguration for Virtex II Pro FPGAs is explored, while in [PJ06] an extension of the FPX SW support is presented.

2.3.2 Moraes-Calazans et al.

The Fernando Moraes and Ney Calazans group from the Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brasil, has defined and developed a 1D, bus based reconfigurable system, briefly described in this subsection.

A block diagram of the system architecture with two reconfigurable modules is shown on the left side of Figure 2.5. The reconfigurable area, following a 1D distribution, allocates reconfigurable modules called slave cores. Slave cores are interconnected through virtual pins to the fixed area called controller (see left side of Figure 2.5). The controller is built of: i) a master core in charge of controlling the access to the outside of the FPGA, ii) an on-chip communication bus and iii) a bus arbiter. The on-chip bus is physically allocated in the bottom part of the FPGA and implements a bidirectional serial communication (1 bit data transmission). In this case, the serial approach has been selected in order to save area as it is stated in [PdMM⁺02]. Virtual pins, as well as the entire bus are built using TBUFs and TBUFs related routing resources. The amount of TBUFs available in each CLB is reduced (two for Virtex and Virtex II devices), therefore if more lines are included in the bus, the bus area requirements in CLB rows grows, while the reconfigurable modules area gets smaller.

Taking into account the amount of wires that pass through virtual pins from the modules to the controller, that is six per modules, a total of 12 TBUFs are required for the two modules example (see Figure 2.5 right side). As a result, it can be concluded, that at least three CLB rows have been used for the serial bus implementation.

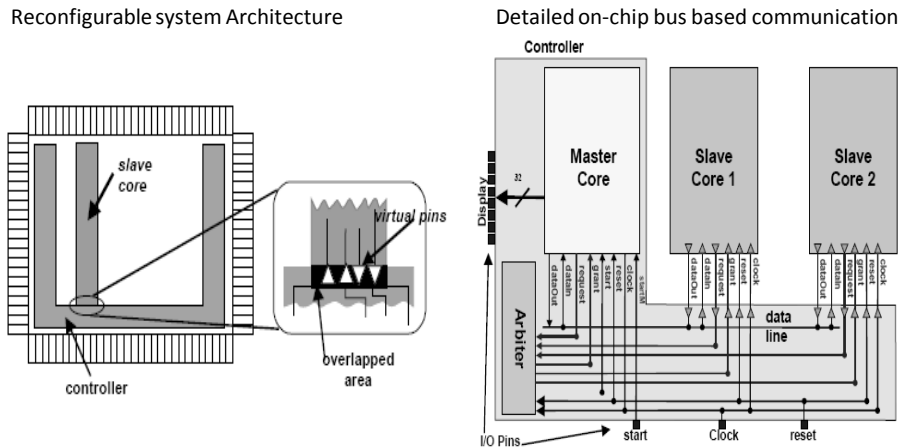


Figure 2.5: Moraes-Calazans et al. reconfigurable system. On the left side the 1D architecture general view and a detailed view of the on-chip communication serial bus on the right side [PdMM⁺02].

The application presented in [PdMM⁺02] as a demonstration of the system is a reconfigurable calculator where each reconfigurable module executes a 32 bits operation (addition/subtraction). In [MMP⁺], the system has been mapped on a Xilinx FPGA, XCV2000 (Virtex) and can be remotely reconfigured. Through a web page some parts of the FPGA configuration file can be modified and loaded in the FPGA. In further publications, like [MSC⁺06] and [MGC⁺07], the bus communication has been replaced by a Network on Chip, but maintaining the 1D architecture model. It is important to remark that the same publications introduce the use of vertical and horizontal communication macros, based on CLB-LUTs instead of TBUFs, discussed later on this Chapter.

2.3.3 Platzner-Walder et al.

The Computer Engineering and Networks Laboratory from the Swiss Federal Institute of Technology Zurich - ETH has developed a reconfigurable platform described in [WNP04], focused on exploring SW systems that make a transparent use of the underlying HW. For this aim, HW reconfiguration control systems, called "HW Operating Systems" (HWOS) are defined.

A 1D architecture has been designed, where the fixed FPGA area implements the HWOS framework by reserving the leftmost and rightmost sides of the FPGA (right and left OS frames in Figure 2.6). The rest of the FPGA, the reconfigurable area, is divided following the Xilinx approach into reconfigurable modules that span the entire FPGA height and are called "slots" [WP04]. A generic view of the system is shown on the left side of Figure 2.6.

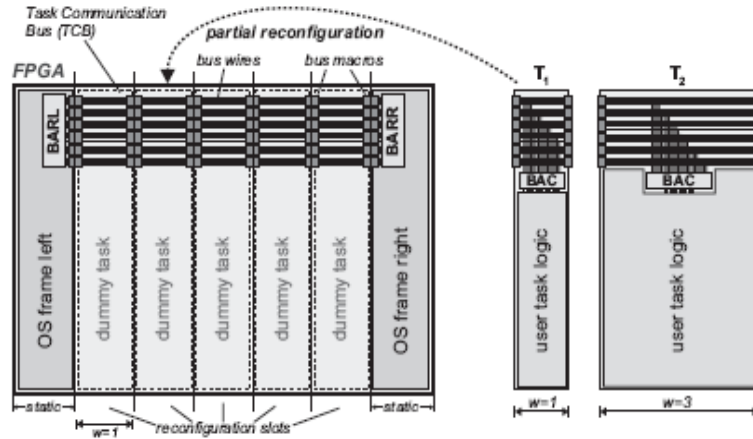


Figure 2.6: Platzner - Walder et al. reconfigurable system, oriented to provide HW support to operating systems (OS). Example with five reconfigurable slots and two OS frame on the FPGA corners and an example of loading different slots of different width [WP04].

To pass data from slots to the OS frames (see Figure 2.6), an on-chip communication structure is defined and allocated on the top part of the FPGA. The communications structure is composed of two independent buses. One is used to transmit data between slots and the right fixed part (right OS frame in Figure 2.6) and the other bus transmits data between slots and the left fixed area (left OS frame in Figure 2.6). Each bus has its own arbiter located on each FPGA fixed part (left and right).

The physical layer of each bus of the communication structure is composed of:

1. Bus macros that are placed in each slot border (left and right) and grant the communication with the neighboring slots.
2. Wires that cross slots from one side to the other hooking bus macros to build bus long wires.

The most interesting characteristic of the described approach is the possibility of grouping slots to allocate bigger HW tasks. Initially all slots are filled with equal dummy tasks that include all the necessary macros and wires for the on-chip communication (see Figure 2.6). Afterwards, if a task wider than one slot (two, three or four slots) is going to be loaded, it does not necessarily need to implement the corresponding to overwritten slots macros (see Figure 2.6 - right side). Implementing the bus macros at the user task borders is enough. After the task execution has finished, the occupied area is again filled with dummy tasks and the initial communication infrastructure re-established by loading again the bus macros. On the other hand, the main drawback of this technique is that it requires freeing the bus during each partial reconfiguration.

According to [WP04], the presented system has been successfully implemented on a Virtex II (XC2V3000) FPGA with slots width of four and eight CLB columns.

2.3.4 Nollet-Mignollet et al.

At IMEC-Belgium, a reconfigurable system for studying aspects related to HWOS, like HW-SW multitasking [MNM⁺04], [NCV⁺03] has been developed. The IMEC work is the first to explore the use of Networks on Chip (NoCs) as a solution for the on-chip communication. The approach will be described more in detail in Chapter 3 that focuses on the use of NoCs in reconfigurable systems. Here, only the designed architecture is briefly reviewed.

The target platform is based on a Virtex II FPGA (XC2V6000) connected to a PDA - iPAQ 3760 with a StrongARM microprocessor. The Xilinx 1D approach has been followed for designing the reconfigurable system and therefore this solution will be included on the summary table presented in the results section 2.4, as an example. With respect to the practical implementation, a 3x3 NoC has been reported in [MNM⁺04].

2.3.5 Hübner-Becker et al.

In [WP04] Hübner-Becker et al., from the University of Karlsruhe, describe a pRTRS based on a 1D architecture with a bus communication.

The fixed area of the system includes a soft-core 32 bit MicroBlaze microprocessor, provided by Xilinx, in charge of controlling the on-chip bus communication, the on-chip ICAP configuration port, as well as, the off-chip communication, see Figure 2.7.

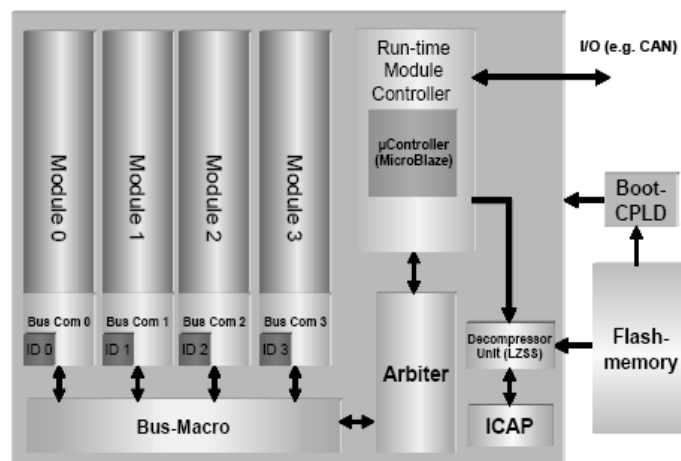


Figure 2.7: Reconfigurable system example with four reconfigurable modules [WP04].

The reconfigurable area is divided into reconfigurable modules, also called, by the author, slots in [MUB04]. Each slot accesses the communication bus through a specific module in charge of module communication control, called BUS-COM (see Figure 2.7). The on-chip bus has been created using communication macros designed with CLB resources. The first time the use of CLBs in communication structures design has been reported was in [MDP02], where they were used as passing-through macros to restrict routing. Differently Hübner et al. offers the use of two, specific macros, that cross the entire FPGA width where CLBs are used as feedthroughs to connect the CLB short wire. As an example, the layout of one of these macros can be seen in Figure 2.8. One of the macros, define as input, transmits signals from slots to the fixed area and the other, called output, transmits signal from the fixed area to slots. The input macro can be seen on the bottom part of Figure 2.9, while the output macro can be seen on the top part. Each macro (referred as BM-Hübner) occupies one CLB row and can route up to eight signals in one direction.

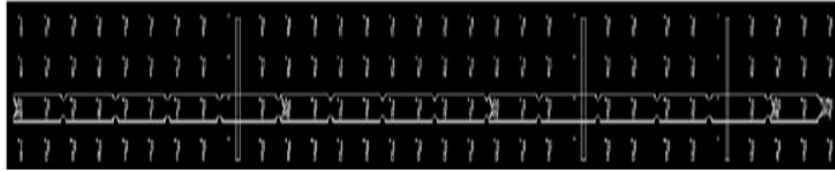


Figure 2.8: Hübner-Becker et al. CLB bus macro (BM-Hübner) layout. Screen shot from the FPGA Editor tool [MUB04].

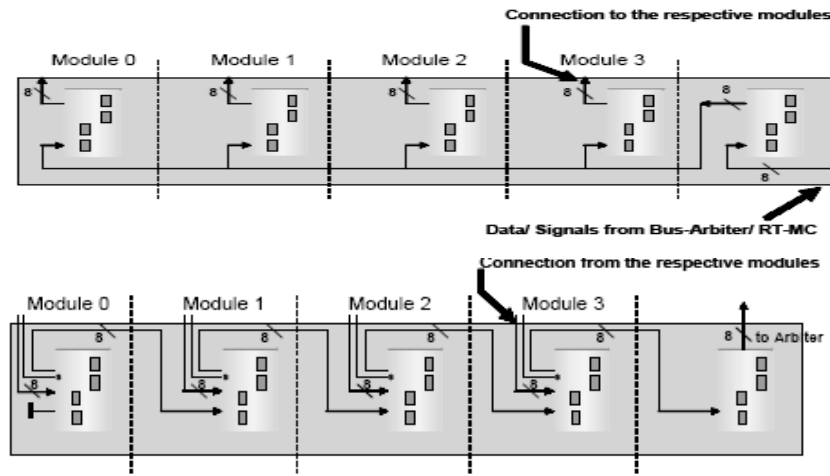


Figure 2.9: Hübner et al. bus macros with highlighted bus accesses. On the top part, the input macro that sends data from slots to the fixed area, and on the bottom part the output macro that sends data from the fixed area to slots [MUB04].

Based on the Hübner et al. approach, a reconfigurable system has been presented in [WP04]. The system is composed of: four slots, a 32 bit bus built with input type macros and a 16 bit bus built with output type macros. Each macro occupies 20 slices and one row, and, therefore, it can be concluded that the on-chip communication infrastructure uses 80 slices and six CLB rows.

As it can be noticed from figures 2.8 and 2.9, the access to the bus is done directly and no special structures for ensuring vertical routing are reported. Also, from the bus layout it could be seen that the used pass-through CLBs (darker squares in the figure) are not regularly distributed, and therefore module relocation is restricted.

It is worthy to highlight that in more recent publications, like [MHB] and [TPB07], the group of the university of Karlsruhe have designed a reconfigurable system following a 2D model approach that includes a NoC for data transmission. This new work will be discussed in Chapter 3.

2.3.6 Erlangen Slot Machine (ESM)

The Erlangen Slot machine (ESM) from the University of Erlangen-Nürnberg, presented in [BMA⁺05a], [BMA⁺05b], [MABT06] and [MTAB07] has been specially created for pRTSR design. The ESM is composed of two boards, one called MotherBoard and the other one called BabyBoard, which provides partial reconfigurability. A schematic view of the Baby-board architecture can be seen in Figure 2.10.

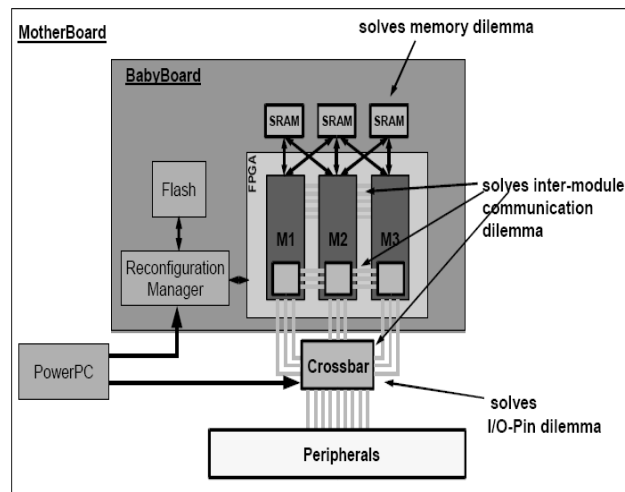


Figure 2.10: Erlangen Slot Machine (ESM) architecture. The Erlangen-Nürnberg reconfigurable systems design platform is composed of two boards. The MotherBoard and the BabyBoar, which provides partial reconfigurability. In the shown example, the system contains three reconfigurable modules and all the existing modules connectivity [MABT06].

The selected FPGA, a Virtex II XC2V6000, has been divided into pieces of four CLB columns width called "Micro-slots". Micro-slots can be, further, grouped to form slots, and this brings some flexibility to the system. Slots, on the FPGA left and right corners,

are reserved for the on-chip communication, while slots in the central part have access to the internal BRAM columns. The remaining slots can access only external memories. The system provides several types of reconfigurable modules connectivity: i) of-chip, using the FPGA bottom IOBs that are connected to a programmable crossbar, a special CPLD (Complex Programmable Logic Device) included in the board (see Figure 2.10), ii) through an SDRAM memory connected to the FPGA top IOBs, and iii) on-chip, including Xilinx bus macros in the system.

The ESM is a relatively flexible platform for building reconfigurable systems. The main, distinguishing, characteristic of the ESM is that it permits, from the architecture point of view, module relocation when the communication is of-chip. However, relocation is not only restricted when on-chip communication is used, but a SW support for relocation has not been reported so far.

2.4 Virtual Architectures for Partial RunTime Reconfigurable Systems. Design Method

Once the existing architectures for partially reconfigurable systems have been analyzed in section 2.1, and the Virtual Architecture (VA) concept has been introduced in section 1.5.2, *a method for designing virtual architectures for partially reconfigurable systems is originally presented in this section.*

The main idea behind this method is to define a set of general steps, which lead to the design of flexible and reliable architectures for reconfigurable systems, which is a requirement for separating the hard core design process from the target system architecture (that is the main thesis approach). *The method permits to design flexible (cores can be allocate/relocate in several FPGA positions, grouping possibilities and specific resources exploitation), reliable (the device is active and stable during reconfiguration) and scalable (independent from the number of slots) systems.*

In contrast to the state of the art it can be said that the commercial solution, the Xilinx approach, does not provides flexibility and does not solve problems related to hard core grouping and relocation. On the other hand, regarding the academic approaches, the other solution that tries cope with FPGA specific resources and bring some flexibility is the ESM. There, slot can be grouped and slots from the middle part of the FPGA can access the on-chip BRAMs. However, the remaining slots do not have access to BRAMs and also relocation, when the communication is done on-chip, is not possible. The Hübner architecture is also based on slots that can be grouped, but relocation and the FPGA specific resources topic is not solved. The same can be said for the Moraes-Calazans et al., where even new cores can be remotely loaded in the FPGA, but core relocation from the architecture point of view is not solved. Nevertheless, it is important to remark that the Lockwood-Horta et al. architecture in conjunction with Moraes-Calazans et al., have target core relocation for Virtex based FPGAs previously to the proposed method in this Chapter (first approach published in 2005) and have achieved relocation on simple, with restricted scalability, architectures composed of two independent DHPs modules (the modules are not interconnected). As a result, it could be said that *a general method that targets architecture flexibility, reliability, scalability has not been presented before and is considered an original contribution of this thesis.*

The next subsection deeply describes the method general steps that have been identified and are summarized in Figure 2.11. A set of design guidelines, considered part of the method, that have to be followed during reconfigurable systems design, and a slot definition can be found in the second subsection where the method steps will be generally applied to two Xilinx FPGAs families (Virtex II and Virtex II Pro). In the last subsection, a practical application of the method steps and the defined guidelines to build two 1D bus based reconfigurable systems for two Virtex II FPGAs can be found.

2.4.1 Method Goal and General Steps

The main characteristics to be covered by a virtual architecture to bring the mentioned flexibility, reliability and scalability are listed below:

1. **Core Relocation.** Relocation is very important because it leads to memory savings and simplifies the SW control system. Relocation permits to keep a unique image of the hard core and allocate it at runtime at the proper position in the virtual architecture. Otherwise, as much images of the hard core as possible slot positions have to be kept in memory.
2. The possibility of *loading hard cores of any size* in the system that also brings flexibility.
3. The architecture has to permit the exploitation of as much FPGA specific resources as possible (high use of the on-chip resources).
4. The internal communication has to permit the system to be scalable.
5. To permits non intrusive partial runtime reconfiguration. This means that the on-chip communication infrastructure has to be kept active during reconfiguration. This is not a trivial task because usually the communication infrastructure is spread across the chip.

The design steps proposed in this subsection along with the guidelines presented in the next subsection, more specific to Xilinx FPGAs, aim to lead to the design of reconfigurable systems VAs that have the characteristics previously listed. A general view of the method design steps can be found in Figure 2.11 and its description is included next:

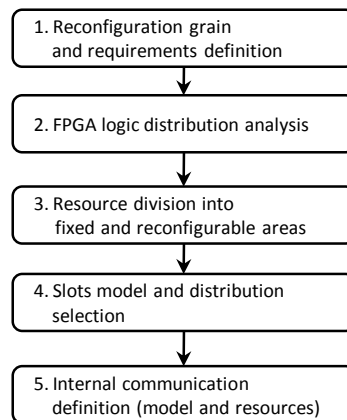


Figure 2.11: Virtual architectures for partial runtime reconfigurable systems - design method general steps.

1. The first step is *to defined the requirements* to be covered by the reconfigurable system. Regarding VAs, it is important to define the type and the granularity of the reconfiguration or reconfigurations that will be supported. For systems with

fine grain, Intra-Module, reconfiguration (some LUTs, single BRAMs and/or some FFs) there is no need of defining a special architecture. The restrictions required to successfully perform a partial reconfiguration, to know the position of the logic to be modified, can be setup during hard core design. This type of reconfiguration will be called hard core adaptation or small bit manipulation (the Xilinx term). On the other hand, VAs are needed for medium and/or coarse grain reconfigurations of Intra-Module or Inter-Module type.

2. Once a partially reconfigurable FPGA that covers the defined area requirements and reconfiguration method(s) has been selected, the FPGA logic distribution has to be analyzed. The aim at this point is to identify the specific *elements distribution homogeneity*. Specific elements could be: embedded memories, DSP blocks, multipliers, configuration ports, clock controller, ADCs, etc.
3. The next step is to define the *FPGA internal resource division* into fixed and reconfigurable area. The main goal in this step is to select such distribution that the resulting reconfigurable area is as homogeneous as possible. All elements that have a non regular distribution should be assigned to the fixed FPGA area (like the ICAP port in Xilinx FPGAs). For instance, if the IOBs distribution across the die is regular, then IOBs can be assigned to the reconfigurable area. In the opposite case, they have to be part of the fixed area and to be accessed using macro structures.
4. Once the fixed/reconfigurable area resource division has been defined, the next step is *to divide the reconfigurable area into* homogeneous reconfigurable modules, called *slots* that allocate hard cores. Each slot has to guarantee hard cores access to the maximum amount of FPGA specific resources, as well as to the on-chip communication infrastructure. If it is foreseen that all hard cores in the final application require access to the same FPGA embedded resources, like multipliers, then this resource, if it is possible and homogeneity is preserved, will be considered as part of each slot. As it can be noticed from the state of the art, the reconfigurable area can be divided into slots following two models: one dimensional (1D) and two dimensional (2D).
5. The last step, but maybe the most important, is to *define the internal communication of the reconfigurable system*. The selection of the communication scheme depends on the required system flexibility and scalability. For simple systems, where there is no need of hard core relocation and the amount of slots is kept low (one or two), simple direct connections are the best options (like in the Xilinx approach). On the other hand for robust and flexible systems, bus (like the Hübner et al. approach) and NoC type internal communication schemes are more suitable. Anyhow, independently from the communication scheme, if flexibility and reliability are targeted, the underlying physical implementation layer has to be prepared for that. The communication structure has to be localized in an FPGA area that is reserved for the internal communication. This permits to keep it active during reconfiguration, even when relocation is being performed. Also, the occupied area has to be kept as low as possible and access to the communication infrastructure has to be guaranteed for all possible slot positions. Another, important, aspect of the physical implementation is to carefully select the communication interconnection building elements and the amount of these

elements to be used, because this will define the interconnection delay, the data transfer rates and the required area. This selection depends on the target FPGA family and will be discussed with an example in the next subsection.

2.4.2 Method Guidelines for Xilinx FPGAs

Next, a set of guidelines for reconfigurable systems design, along with a slot definition are proposed following the method steps and using as target two Xilinx FPGAs families.

2.4.2.1 System Reconfigurability Requirements and Target FPGAs

During this general application of the method, the system requirement will be that the system has to support Intra and Inter-Module reconfiguration of medium and coarse grain. As it has been mentioned several times, nowadays, the Xilinx FPGAs are the most feasible solution for designing reconfigurable systems, and the selected FPGA families in this case are the Virtex II and Virtex II Pro family FPGAs.

2.4.2.2 Logic Distribution Analysis

FPGAs as general can be seen as a sea of logic elements with interleaved, specific, elements. For Virtex II and Virtex II Pro FPGAs, these elements are: memory blocks (BRAMs), multipliers (MULs) and/or microprocessors (PPCs), as it can be seen in Figure 2.12 where an FPGA Editor layout of a device from each FPGA family can be found (for Virtex II on the left side and for Virtex II Pro on the right side). Regarding BRAMs and MULs, their distribution is quite regular as there is one column of these elements every certain amount of CLBs. On the contrary, IOBs distribution is not regular. The amount of IOBs at the top and the bottom of a CLB column is not always the same. On the other hand, the elements that are not wide spread, like PPCs in Virtex II Pro for example, are considered as not regular although their position in the FPGA is symmetrical.

2.4.2.3 Resource Division

After the FPGAs homogeneity has been analyzed, in Figure 2.13, a general resource division in fixed and reconfigurable area can be found. The Virtex II resource division is shown on the left side, and for Virtex II Pro family on the right side of Figure 2.13.

For achieving homogeneity in the reconfigurable area, the fixed area of the virtual architecture that has to be defined includes:

1. The fixed area uses the leftmost and rightmost FPGA sides and for Virtex II, also the middle CLB columns, because there is an irregularity in the amount of CLB columns in the die center. Anyway, this area can be used to implement modules in charge of internal communication control, as well as external device communication. All the remaining logic is reserved for the reconfigurable area that is divided into slots.

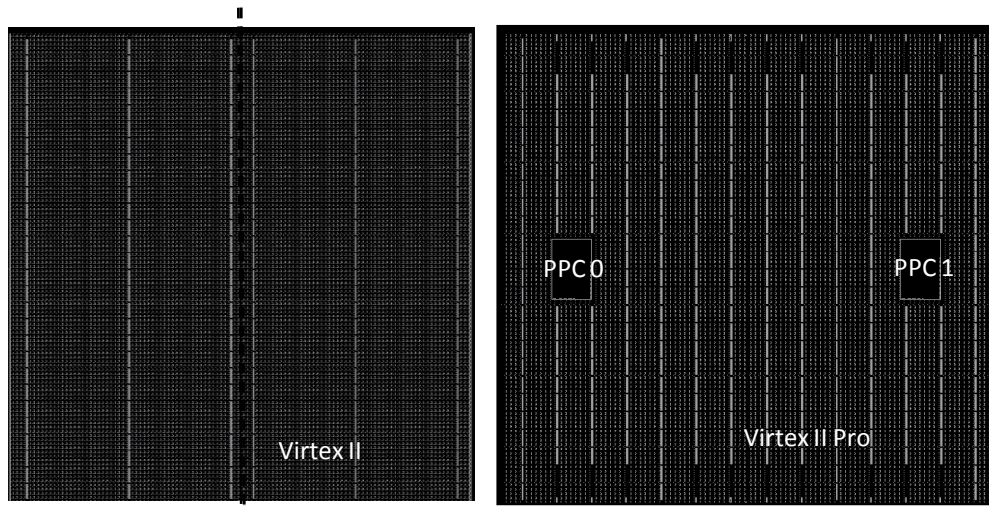


Figure 2.12: FPGA Editor layout. For a Virtex II FPGA on the left and for a Virtex II Pro on the right.

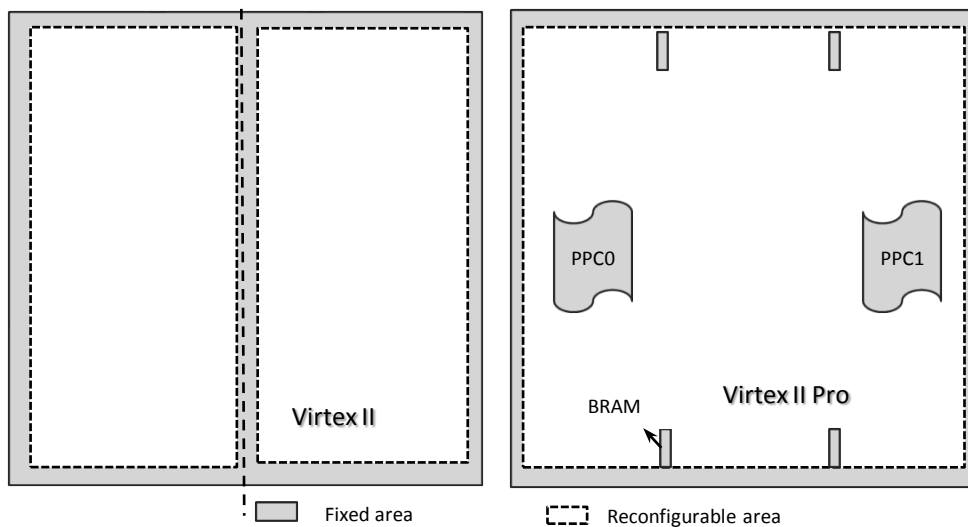


Figure 2.13: Resource Division for Virtex II FPGAs on fixed and reconfigurable area on the left, and for Virtex II Pro on the right.

2. As there is no homogeneity on the IOBs distribution across the die, the IOBs on the upper part of the FPGA are assigned to the fixed area.

For Virtex II Pro FPGAs, a slightly more complex resource division is needed. In this case apart from the IOBs and the leftmost and rightmost CLB columns, in order to preserve regularity, the fixed area also includes:

1. The embedded PPCs along with some surrounding CLBs and/or BRAMs. The amount of CLBs and BRAMs depends on the selected model for the VA and the amount of area needed for the fixed area logic implementation. As it has already been mentioned, the two possible models are 1D and 2D. For instance, for the 1D model all the CLBs that are on the PPC upper and bottom part have to be part of the fixed area.
2. Some memory blocks. The amount of BRAMs is not equal for all BRAM columns. Therefore for achieving homogeneity (the number of BRAMs in BRAM columns has to be equal), some BRAMs are prohibited for being used and are included on the fixed area, as it can be noticed from the right side of Figure 2.13.

2.4.2.4 Slot Definition and in 1D and/or 2D Models

The slots definition, originally presented next, leads to the design of flexible VAs.

1. All slots in a reconfigurable system have to be equal in terms of shape, composition and size. In the case of Virtex II and Virtex II Pro FPGAs, for achieving maximum homogeneity, slots can be based only on CLB columns. BRAMs, MULs, DCMs and other specific elements are not considered part of a slot, they are separate elements.
2. Slot boundaries have to be defined such that hard cores allocated in them have access to on-chip specific resources, BRAMS and MULs.
3. A slot does not span the entire FPGA height or all the FPGA rows in a reconfigurable region (for Virtex 4 and Virtex 5). Slot boundaries are restricted by the communication structure borders. This facilitates hard core relocation.
4. A slot accesses the on-chip communication structure only through specially defined for that access points. Access points have to be placed in the same position relative to the slot borders, for all slots. This permits relocations.

With the previously presented slot definition, architectures with 1D and 2D slots distributions have been defined in the next paragraphs for Virtex II and Virtex II Pro FPGA families.

In order to divide a reconfigurable area into slots, first some parameters related to the FPGA capabilities, in the context of reconfigurable systems, have been defined in Table 2.1 and will be further used.

The amount of slots that can be defined in a reconfigurable area and the slots size (SlotSize) depends on the target FPGA available logic resources. Therefore, the slots size is defined by slots width - SlotWidth and slots height - SlotHeight and are measured in CLB Columns and CLB rows respectively.

Usually, in virtual architectures with one dimensional resource division, the on-chip communication infrastructure is allocated in the FPGA top or bottom part. Example 1D based architectures, where the communication structure occupies the bottom part of the FPGA, can be seen for Virtex II FPGAs on the left side of Figure 2.14, and for Virtex II Pro

Table 2.1: FPGA Virtual Architecture Definition Parameters

Parameter	Definition
chipCLBCols	Number of device CLB Columns
chipCLBRows	Number of device CLB Rows
chipRAMCols	Number of device BRAM column
fixedCLBCols	Number of CLB columns occupied by the fixed area
fixedCLBRows	Number of CLB rows occupied by the communication structure (for 2D VAs this value is per slot row)
fixedRAMCols	Number of BRAM columns occupied by the fixed area
chipPPCs	Number of embedded microprocessor in the device
N	Number of CLB columns between two BRAM columns
S	Number of slots in the virtual architecture
SClos	Number of slot columns in the virtual architecture
SRows	Number of slot rows in the virtual architecture

FPGAs on the right side. It is worth pointing out that with Virtex II Pro devices a more regular structure, where there is a BRAM column next to each slots, can be achieved. However, due to the PPCs, there are at least twelve CLB columns that are additionally assigned to the fixed area, see Figure 2.14. On the contrary, for virtual architectures that follow a two dimensional resource division, the communication structure is allocated along the FPGA device. There are two communication channels defined per slot row, one on the slot top and other on the bottom, permitting each slot to be connected to as much slots as possible. A general view of a 2D model can be found on the left side of Figure 2.15, while its general mapping on a Virtex II with two rows can be seen on the right side.

A set of relations have been identified, based on the described slots distribution and using the definitions included in Table 2.1. These relations permit to calculate different combinations of number of slots (S) to slotSize ($S/\text{slotSize}$), which can be further used during reconfigurable system design for selecting the most appropriate combinations, depending on the target application. Relations for Virtex II FPGAs, valid for both models can be found in Table 2.2, while for Virtex II Pro they can be found in Table 2.3.

Based on the relations presented in tables 2.2 and 2.3, some $S/\text{slotSize}$ combinations have been calculated and can be found for Virtex II in Table 2.4 for 1D slot distribution, and in Table 2.5 for the 2D distribution. For Virtex II Pro, the $S/\text{slotSize}$ combinations can be found for 1D and 2D in tables 2.6 and 2.7 respectively. For Virtex II FPGAs, slot widths that result in slots sizes bigger than 70 CLBs have been taken into account and have been included in the tables. On the contrary, for Virtex II Pro FPGAs, the selected slot width is constant, $\text{slotWidth} = N$, because these FPGAs have more regular BRAM distribution. Additionally, depending on the selected application it could be recommended to include some BRAMs in each slot if the slots equality is preserved.

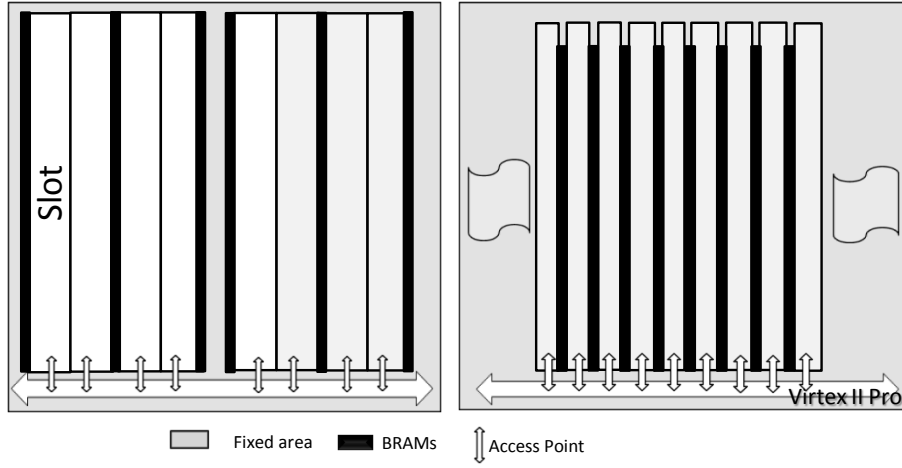


Figure 2.14: One dimensional general virtual architecture for Virtex II FPGA families on the left and for Virtex II Pro on the right.

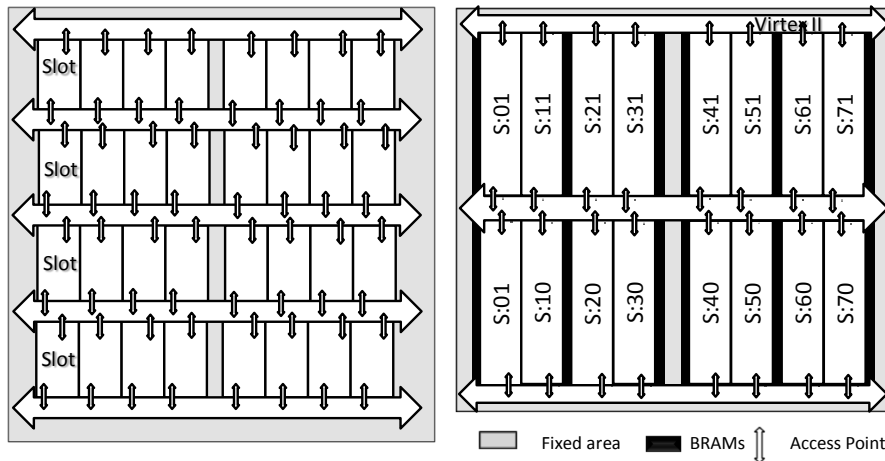


Figure 2.15: Two dimensional virtual architecture general view on the left and an example mapping for Virtex II on the right.

Regarding the communication structure occupied area, for the 1D approach it has been defined to be four rows ($\text{fixedCLBRows} = 4$). It is important to notice that the communication structure area is reserved for the on-chip communication implementation. This permits to achieve hard core relocation and stable reconfiguration (two of the defined requirement). For the 2D slot distribution tables 2.6 and 2.7, the communication structure occupied area has been defined to be eight CLB rows, ($\text{fixedCLBRows} = 8$), four for the bottom slot channel and four for the top channel. For

Table 2.2: Virtual architectures design relations for Virtex II FPGAs.

Parameter	Value
N	$\frac{chipCLBCols - fixedCLBCols - 2}{chipRAMCols - fixedRAMCols}$
SClos	$\frac{N * (chipRAMCols - 2)}{SlotWidth}$, $4 \geq SlotWidth \leq N$
SRows	2, 4, 8...
S	$SClos * SRows$
slotWidth	4, 6, 8...
slotHeight	$\frac{chipCLBRows}{SRows} - fixedCLBRows$
fixedCLBRows	4, 8..., chipCLBRows

Table 2.3: Virtual architectures design relations for Virtex II Pro FPGAs.

Parameter	Value
N	6
SClos	$\frac{N * (chipRAMCols - 2ChipPPCs - 1)}{SlotWidth}$, for 1D
SClos	$\frac{N * (chipRAMCols - 1)}{SlotWidth}$, for 2D
SRows	2, 4, 8...
S	$SClos * SRows$
slotWidth	N, 2N, 4N...
slotHeight	$\frac{chipCLBRows}{SRows} - fixedCLBRows$
fixedCLBRows	4, 8..., chipCLBRows

example, following a 1D model, the possible amount of slots that can be defined in an virtual architectures that target an XC2V3000 FPGA is 4, 8 or 12 ($S = 4|8|12$ in Table 2.4). The slots width related to each possible amount of slots is 12, 6, or 4 CLBs respectively ($SlotWidth = 12|6|4$ in Table 2.4). The slots height is 108 CLBs and therefore the resulting slot size in CLBs is 720, 360 or 240 respectively ($SlotSize = 720|360|240$ in Table 2.4). The same system is used all the FPGAs and for Table 2.5.

Table 2.4: Virtual architecture slots distributions for Virtex II - 1D Model

Parameter	Device						
	XC2V1000	XC2V1500	XC2V2000	XC2V3000	XC2V4000	XC2V6000	XC2V8000
LC	1M	1.5M	2M	3M	4M	6M	8M
chipCLBRows XchipCLBCols	40X32	48X40	56X48	64X56	80X72	96X88	112X104
chipBRAMCols	4	4	4	6	6	6	6
BRAMs per Col	10	12	14	16	20	24	28
N	12	16	20	12	16	20	24
S	2 4	2 4 8	2 4 10	4 8 12	4 8 16	4 8 20	4 8 16 48
SlotWidht	12 6	16 8 4	20 10 4	12 6 4	16 8 4	20 10 4	24 12 6 4
SlotHeight	36	44	52	60	76	92	108
SlotSize(CLBs)	432 216	704 352 176	1040 520 208	720 360 240	1216 608 304	1840 920 368	2592 1296 648 432
LC Logic Cell							

Table 2.5: Virtual architecture slots distributions for Virtex II - 2D Model

Parameter	Device						
	XC2V1000	XC2V1500	XC2V2000	XC2V3000	XC2V4000	XC2V6000	XC2V8000
LC	1M	1.5M	2M	3M	4M	6M	8M
chipCLBRows XchipCLBCols	40X32	48X40	56X48	64X56	80X72	96X88	112X104
BRAMCols	4	4	4	6	6	6	6
BRAMs per Col	10	12	14	16	20	24	28
N	12	16	20	12	16	20	24
S	2 4	2 4	2 4	4 8 12	4 8 16	4 8	4 8 16 48
SRows	2	2	2	2	4	4	4
SlotWidht	12 6	16 8	20 10	12 6 4	16 8	20 10	24 12 6 4
SlotHeight	12	16	20	24	12	16	20
SlotSize(CLBs)	144 72	256 128	400 200	288 144 96	192 96	320 160	480 240 120 80
LC Logic Cell							

Table 2.6: Virtual architecture slots distributions for Virtex II Pro - 1D Model

Parameter	Device						
	XC2VP7	XC2VP20	XC2VP30	XC2VP40	XC2VP50	XC2VP70	XC2VP100
LC	11M	20M	30M	43M	53M	74M	99M
chipCLBRows XchipCLBCols	40X34	56X46	80X46	88X58	88X70	104X82	120X94
BRAMCols	6	8	8	10	12	14	16
BRAMs per Col	8	12	18	10	12	14	16
PPCs	1	2	2	2	2	2	2
N	6	6	6	6	6	6	6
S	3	3	3	5	7	9	11
SlotWidht	6	6	6	6	6	6	6
SlotHeight	36	54	76	84	84	100	114
SlotSize(CLBs)	216	324	456	504	504	600	684
LC Logic Cell							

Table 2.7: Virtual architecture slots distributions for Virtex II Pro - 2D Model

Parameter	Device						
	XC2VP7	XC2VP20	XC2VP30	XC2VP40	XC2VP50	XC2VP70	XC2VP100
LC	11M	20M	30M	43M	53M	74M	99M
chipCLBRows XchipCLBCols	40X34	56X46	80X46	88X58	88X70	104X82	120X94
BRAMCols	6	8	8	10	12	14	16
BRAMs per Col	8	12	18	10	12	14	16
PPCs	1	2	2	2	2	2	2
N	6	6	6	6	6	6	6
S	4	6	6	8	10	12	14
SRows	2	2	2	2	4	4	4
SlotWidht	6	6	6	6	6	6	6
SlotHeight	12	20	32	14	14	18	22
SlotSize	72	120	192	84	84	108	132
LC Logic Cell							

2.4.2.5 Communication Structure: Definition and Building

The communication structure and the interconnections building elements definition is the last step of the proposed method, but maybe the most important, as this physical layer defines the limits of the entire on-chip communication reliability, flexibility and scalability.

The VA method covers two main aspects of the communication structures: i) the definition of the communication scheme interconnections and ii) the selection of the interconnections building elements (macro structures) from the available resources in the FPGA.

Along the next paragraphs, first the available interconnections macros building resources in Virtex II and Virtex II Pro FPGAs are discussed, then some communications macro structures are described, and at the end of the section, they are compared highlighting their advantages and disadvantages.

The available communication macros building resources in Virtex II and Virtex II Pro FPGA can be divided into: i) TBUFs based resources, that include tri-state buffers and the related tri-state routing lines (see Figure 2.16) and ii) CLBs based resources, CLBs Look Up Tables (CLB-LUTs) and the CLB routing resources (see Figure 2.17) .

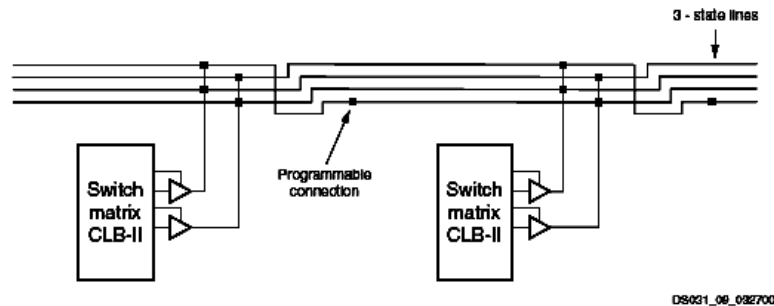


Figure 2.16: TBUFs and TBUFs Routing Resources [Xil04].

There are two TBUFs per CLB and four TBUFs routing wires per CLB row. Each buffer wire has programmable connections allocated every forth CLB column (see Figure 2.16). These programmable connections can be used to concatenate the routing wires to design long wires that span the entire FPGA width, like the BM-TBUF-L discussed later and shown in Figure 2.18. Differently, the BM-TBUF-Xilinx described in the introduction part connects neighboring modules. It is also important to mention that not all the FPGAs have these buffers. For instance there are no buffers in Spartan 3 FPGAs and therefore it is not possible to use this type of macros.

The second type of available resources are the CLB based. There are several types of CLB routing wires, shown in Figure 2.17, that are briefly described next, highlighting the advantages and disadvantages of using each wire type.

- Long lines. Vertical and horizontal long lines span the full height and width

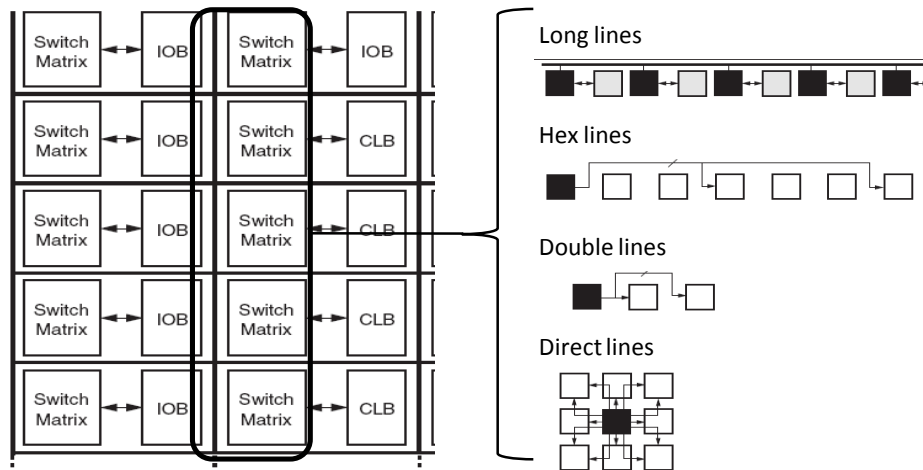


Figure 2.17: Routing resources available in Virtex II/Pro FPGAs [Xil04].

of the device. The access to these lines through the device switch matrices is complicated and therefore they have been excluded as an option for building on-chip interconnections.

- Hex lines route signals to every third or sixth block away in all four directions. Hex-line signals can be accessed either at the end-points or at the mid-point (three blocks from the source). This type of wires is suitable for building long or short on-chip interconnection macros.
- Double lines route signals to every first or second block away in all four directions. These lines are suitable for building short on-chip interconnection macros for neighboring slots connections.

After the possible communication macros building elements have been described, in the next paragraphs, the on-chip interconnections that can be build, shown in Figure 2.18, are reviewed.

Interconnections can be generally classified in two types: i) point to point interconnections (P2P) that are direct links between modules (slots) and ii) point to multi-point interconnections (P2M), links that connect to more than two modules (slots).

2.4.2.6 Point to Point Interconnections (P2P)

From the physical point of view, two different P2P interconnections have been identified: i) short interconnections, that communicate two neighboring items, slots, fixed area and/or IOBs, (see Figure 2.18) and ii) long interconnections, that communicate two non neighboring items and its minimal width is defined by the crossed slots width (see the bottom part of Figure 2.18).

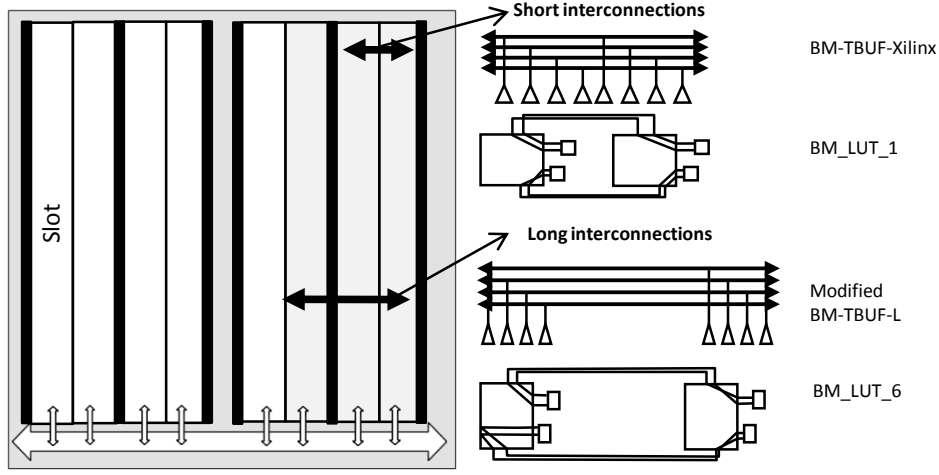


Figure 2.18: Point to Point Interconnection (P2P) example. A short interconnection is shown on the top part of the figure and a long interconnection is shown in the bottom part.

Furthermore, taking into account the building elements of the P2P interconnection macros, they could be: i) buffer based (BM-TBUF), like the Xilinx BM-TBUF-Xilinx or the modified BM-TBUF-L long interconnection macros (see Figure 2.18) and ii) CLB based macros, called BM-LUT, because CLB LUTs are used for pass-through (see Figure 2.18).

Buffer based macros have been already described in the introduction section 2.1 where the BM-TBUF-Xilinx was discussed. Here these macros have been slightly modified to design a long interconnections macro (BM-TBUF-L) shown on Figure 2.18 that can interconnect two non neighboring modules.

On the other hand, for building short BM-LUTs, CLB double lines are appropriate, while Hex lines are more suitable for long interconnections as it can be seen in Figure 2.19, where an FPGA Editor picture of a short macro can be seen on the top part and a long macro on the bottom part. Each BM-LUT requires one CLBs (composed of four Slices) from each item involved in the interconnection and can pass up to eight signals from one item to another. Unlike TBUFs, two different types of CLB-LUT macros can be built, vertical and horizontal.

For the purposes of this thesis, a set of vertical and horizontal short and long macros has been designed. All designed macros are bidirectional and pass 4 signals from left/up to right/down. The distance between the two macros end points, measured in CLB columns is the number that appears at the end of the macro structure name. For example, BM-LUT-6 (see Figure 2.19), indicates that the macro connects two elements that are at 6 CLB columns distance. CLB macros of 1, 4 ,6 and 8 CLB columns distance have been designed and used for the reconfigurable systems presented in along this thesis.

Normally, single distance BM-LUTs (BM-LUT-1) should be used for neighboring slots

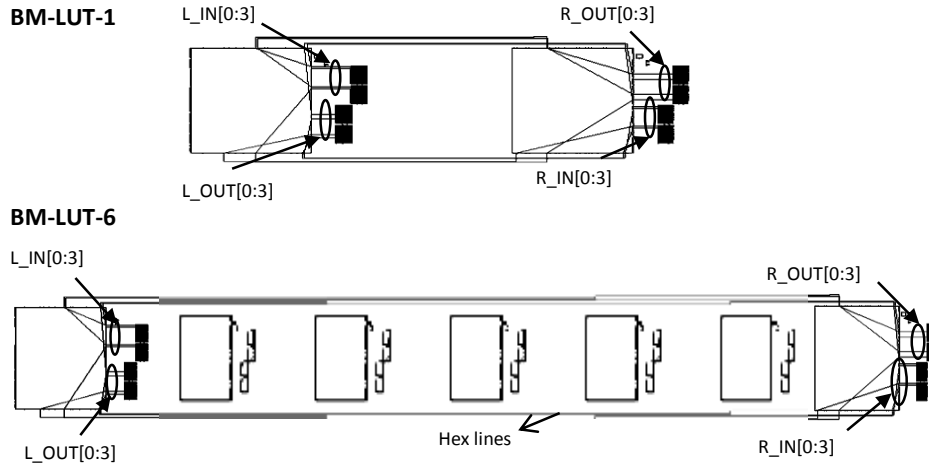


Figure 2.19: Two horizontal CLB based communication macros interconnection (BM-LUTs) FPGA Editor images. One short macro that communicates two neighboring elements (BM-LUT-1) is shown on the top part and on the bottom, a long macro where one of the interconnection points stands at six CLBs distance (BM-LUT-6).

connection, but tests have shown that better results for neighboring slots connections have been achieved with the six CLB columns distance macros (CLB-LUT-6), because there is more distance from the macro to the slot border and thus the border crossing problem is reduced (most of the signals are routed within the defined slot borders). It is very important to clarify that if there is a single wire crossing the slot border, relocation is not possible. Long macros can also be nested to allocate several in a single row.

To summarize, CLB and buffer based P2P macros features can be found in Table 2.8 where the advantages and disadvantages of using each solution are included.

Finally, it is important to remark that there are several CLB based macros available in the state of the art, including some provided by Xilinx with the Plan Ahead tool. But they are subsequent to the beginning of this thesis work and therefore have not been used for reconfigurable systems design. As an example it is noteworthy to mention that some macros include control signals (like an enable signal) [MSC⁺06].

2.4.2.7 Point to Multi Point Interconnection (P2M)

For point to multi point interconnection (P2M), both CLBs and TBUFs related resources can be used. These wires, that cross the entire FPGA height or width, are built by linking either CLBs HEX lines (see Figure 2.17) or TBUF routing lines, see Figure 2.16. CLBs based routing resources are suitable for building shared wires that transmit signals from the fixed area to slots, while TBUFs based routing resources are suitable for bidirectional wires, where the direction can be defined depending on the loaded HW core. Independently from the interconnection type, each macro structure (P2P or

Table 2.8: P2P buffer and CLB based communication macros features comparison

	BM-TBUF	BM-LUT
Used Logic	Tri-state Buffers	CLB-LUTs
Comm. wires per row	Up to four	Eight per CLB used
Comm. direction	Defined by the communication protocol	Defined at macro design time
Advantages	1) More generic 2) Does not uses logic resources	1) Easier to control 2) Provide more comm. wires 3) Can be nested
Disadvantages	1) Restricts module width to 4 CLBs 2) Some problems are reported when all wires are defined in the same direction 3) Some FPGAs do not have TBUFs (Spartan 3).	CLBs have more delay that TBUFs

P2M) is design manually using the FPGA Editor tool, or it can be written in an xdl (Xilinx Description Language) file that is a text description of each macro used resource configuration. The resulting macro structures could be an .nmc file (FPGA Editor) that is a relative placement macro file or an .xdl file.

2.4.3 Virtual Architectures Design Method Result. Definition files

The result of the virtual architecture design method is a virtual architecture framework that includes:

1. **System template**, a vhdl file that defines the system architecture and the design framework. The file defines the slot interface, the communication channels size, the number of used macros and their connection to slots (access points). A pseudo code of the system template can be seen in Table 2.9. The template definition permits to easily integrate new hard cores to a system or to design individual hard cores. By default the system template is empty and therefore empty hard core are mapped to all slots.
2. **Hard core template**, a vhdl file that defines the hard core interface. At this level, the system architecture is fully independent of the on-chip communication protocol and the cores to be instantiated in the template. The hard core interface is standard and core independent, as it does not includes any kind of communication protocol definition. It is connected to the slot interface (access points) in the higher architecture level (in the system template). A pseudo code of a slot template can be seen in Table 2.10.

Table 2.9: Systems template pseudo code

Systems Template

```

entity reconf_system is port (
Reconf_system_inputs
Reconf_system_outputs ); end;

component hard_core_bounding_box is port (
hard_core_bounding_box_interface_inputs ;
hard_core_bounding_box_outputs ); end component;

for (num_macros.types.to.be.used), write
{ component macro_type is
( macro_inputs;
macro_outputs ) end component; }

for_num_of_on-chip_communication_macros
{ signal local_signals; }

architecture RTL of reconf_system is
begin
— port maps
Fixed : fixed_core_design port map (
fixed_core_interface => local_signals );

for num_of_slots, write {
Slot_X: hard_core_bounding_box port map (
hard_core_bounding_box => local_signals ); }

for_num_of_on-chip_communication_macros write {
CS_X : [macro_type] port map (
macro_inputs => local_signals,
macro_outputs => local_signals, ); }

```

Table 2.10: Hard core template pseudo code

hard core Template

```

entity hard_core_bounding_box is
port (
standard_interface_inputs;
standard_interface_outputs );
end;

component core_name is
port ();
end component; }

architecture RTL of hard_core_bounding_box is

begin
slot_standard_IF: slot_standard_IF port map ();
for num. of cores in slot, write {
core: core_name port map (); }

```

3. Macro structure related files (.mnc and .xdl files) that define the internal communication structure macros.
4. The user constraint file (.ucf) that defines the architecture slots distribution and the position of all the macro structures.

The defined template based virtual architecture framework, along with the hard core design flow presented in Chapter 4, permits to design hard cores without knowing the reconfigurable system architecture details (routing, free slots, etc). For this aim, simply a similar VA has to be selected (with enough slots, suitable slot area and number interconnection wires), at it will be shown further in this thesis.

2.4.4 Practical Application to 1D Partial Runtime Reconfigurable Systems

The design method and guidelines, previously presented in this Chapter, have been used for building three runtime reconfigurable systems. Two of them are one dimensional virtual architectures that use bus communication and are described in this section, while the third one is a NoC based reconfigurable system with a two dimensional virtual architecture that is the main topic of Chapter 3.

In order to design the reconfigurable systems, two general one dimensional slot distributions for two different Virtex II FPGA will be selected from Table 2.4. By selecting the slot distribution, the first four steps of the presented method are covered. For the last step, related to the on-chip interconnection, all the knowledge present in the previous section will be used to design bus communication structures that occupy less area and have better performance than the state of the art, as it will be seen in the results section of this Chapter. Also, the designed reconfigurable systems have been tested in terms of stability during reconfiguration and hard core relocation (reliability and flexibility goals).

The first system, described next, is for an XC2V1000 FPGA and the second one, described afterwards, is for an XC2V3000.

2.4.4.1 Bus Based Reconfigurable System for an XC2V1000 FPGA (Bus-v1)

This system is a first approach to achieve hard core relocation and the integration of the on-chip communications in the smallest possible area.

As it has been mentioned, the first four steps of the method are common for an entire FPGA family and have already been presented in section 2.4.2. Here, a slot distribution will be selected and the last point of the method, related to the on-chip communication is described below.

The selected slot distribution from Table 2.4 for the target FPGA, XC2V1000, is the one composed of six slots of four CLB columns width. However, the two rightmost slots have been reserved for on-chip bus arbitration. The remaining four slots have been used for hard core allocation tests.

A piece of the designed bus communication structure can be seen in Figure 2.20. The on-chip bus communication has been based on long P2M interconnection (see Figure 2.20). For achieving resource integration and thus lower area overhead, the bus has been composed of two wires types, built by different FPGA resources: i) based on long BM-TBUFs, four bidirectional wires used as data lines have been implemented and ii) eight P2M unidirectional wires from the fixed area to slots using long CLB interconnections. The total amount of connection wires is twelve; however the communication structure needs only a single FPGA row resulting in higher integration. Differently, if the same amount of interconnections were implemented only with TBUFs, three rows would be needed.

Access points are key elements that permit to use the on-chip communication structure and guarantee relocation. They are designed using vertical short P2P CLB macros that have been slightly modified (see Figure 2.20). The CLB on one side of the interconnection has been hooked to the communication structure wires. Additionally, the FPGA bottom IOBs has been connected to the communication structure for validation purposes (reliability test).



Figure 2.20: Bus communication structure for an XC2V1000 (Bus-v1) built by long CLBs and buffer based macros.

All the files needed for the virtual architecture definition: the system and hard core templates, the macro structure files and the user constraint files have been created.

On top of the communication structure, using the architecture template, a bus based synchronous protocol has been defined for testing purposes. The system test IOBs have been connected to a logic analyzer to check data transitions stability during reconfiguration and, a set of relocated (with the tools presented in Chapter 4) hard cores that constantly transmit data to the fixed FPGA area have been loaded consecutively in different slot positions. Initially a core that communicates with fixed area (on the FPGA right side) has been loaded in the leftmost slot resulting in data transmissions along all

the FPGA width. After that, relocated cores have been loaded in different intermediate slots. As a result, the reconfigurable systems have been validated in terms of stability of the on-chip communication during reconfiguration and in terms of flexibility, as relocation tests have been successful.

2.4.4.2 Bus Based Reconfigurable System for an XC2V3000 FPGA (Bus-v2)

This, second, testing reconfigurable system main goal is to check the scalability of the solution by targeting another bigger FPGA (XC2V3000), increasing the number of bus wires and adding long P2P wires to the bus. Also, the number of required pass-through CLBs is reduced, as well as the possibility of border crossing wires, compared to the previous, Bus-v1, system. Again, a slot distribution will be selected from Table 2.4 and the last point of the method is described next. A section of the Bus-v2 communication structure can be seen in Figure 2.21.

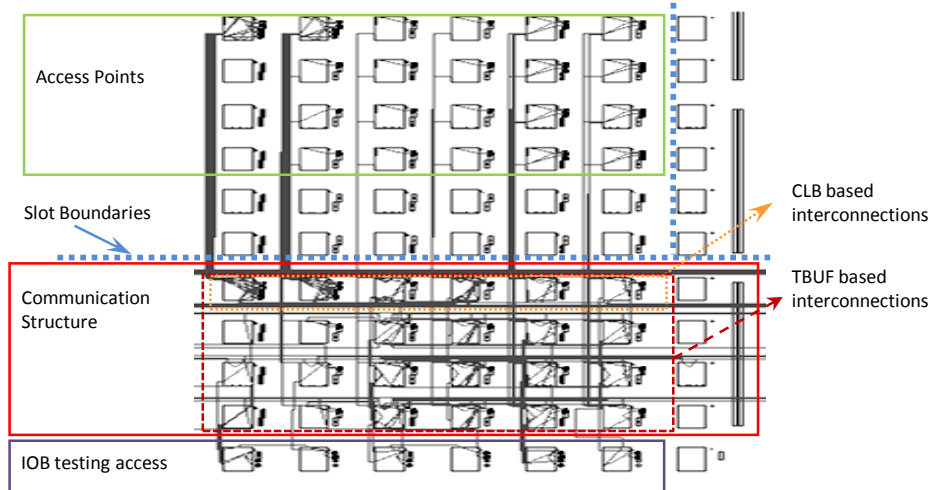


Figure 2.21: Bus communication structure for an XC2V3000 (Bus-v2), built by long CLBs and buffer based macros.

The selected slot distribution from Table 2.4 for the target XC2V3000 FPGA is the one that has eight slots of 6 CLB columns width. Again, the two rightmost slots have been reserved for bus arbitration.

Differently from Bus-v1, here the designed communication infrastructure is composed of three connection types:

1. Sixteen signals based on TBUFs long P2M bidirectional interconnections used for data transfer.
2. Sixteen unidirectional long P2M CLB interconnection wires from the fixed area to slots.

3. Eighteen P2P connections between slots and fixed area used for bus requests.

The total amount of connection wires is 50 and are distributed in four FPGA rows. The TBUFs based interconnections use all the four rows, while all the 34 CLB connections use a single row. This has been possible by interleaving long CLB interconnection (see Figure 2.21). Differently, if the same amount of interconnections were implemented only with the TBUFs, 13 rows would be needed. Access points to the communication structure are designed using slightly modified long P2P vertical CLBs interconnections. Every CLB used as an access point yields up to 8 signals (one per LUT). Connections between the access point and the communication structure are done by vertical CLB routing signals. In the case of TBUF based wires, access point signals are directly connected to TBUF inputs, while for connecting to CLB signal, a CLB LUT is used as a driver (see Figure 2.21). Again, for monitoring the behavior of the wires during partial reconfiguration, some of them have been connected to the IOBs in the bottom part of the FPGA.

Differently from the Bus-v1, access points in Bus-v2 are allocated six CLB rows from the slot border (see Figure 2.21). With this technique, better results in terms of signal routing within the slot boundaries have been achieved. A lot of wires have to be routed to access points, and the Place and Route (PAR) programs use HEX wires that span exactly six CLBs. Therefore with this solution, fewer wires cross the slot borders after the PAR process. Also, the amount of used CLBs have been reduced by eliminating the pass through CLBs (pass through CLBs can be seen in Figure 2.20). Instead of the pass through CLBs, wires have been concatenated simply using CLB switch matrices (see Figure 2.21).

All the files needed for the virtual architecture definition: the system and hard core templates, the macro structure files and the user constraint files have been created.

For testing purposes, on top of the communication structure, a simple synchronous bus protocol that uses 38 of the 50 available wires has been defined. Similarly to the previous reconfigurable system, here also a set of basic partial coarse grain reconfiguration of CLB columns have been performed to validate the system relocation feature.

Finally, it is important to remark that this reconfigurable system has been used as a base for creating a remotely reconfigurable device that will be presented in detail in Chapter 5.

2.5 Results

In Table 2.11, a feature summary of the one dimensional bus based architectures included in the state of the art section 2.1 and the architectures that could be design with the method proposed in this Chapter can be found. The proposed solution also covers 2D architectures, but they are the main topic of Chapter 3 that includes a similar features summary Table. It is important to clarify that in the Table, BM-Xilinx are refereed when the author has not specified the used macro type (CLB or TBUF), but it is clearly based on neighboring communication (Xilinx approach). Differently if data is available, the macro type has been specified (TBUFs or LUT). From all the performed tests and the features summary of the reconfigurable system presented in this Chapter, included

Table 2.11: bus based 1D reconfigurable systems - features summary. Overall comparison of all the approaches presented in the state of the art section of the Chapter and other, based on the proposed in this thesis method.

Group	Device	Macros	Comm.Type	Relocation	Slot grouping
Xilinx	All	BM-TBUFs/LUTs	P2P	No	No
Lockwood	FPX Virtex	No Macros	P2P	restricted	No
Horta et al.	Virtex	BM-TBUFs	Bus	No	No
Moraes et al.	Virtex II	BM-LUTs	Bus	No	No
Platzner et al.	XF Virtex II	BM-Xilinx	BuS	No	No
Nollet et al.	Virtex II	BM-Xilinx	NoC	No	No
Hübener et al.	Virtex II	BM-LUT-Hübner	Bus	No	Yes
ESM	Virtex II	BM-Xilinx	P2P	restricted	yes
Proposal	Virtex II	BM-TBUFs/LUTs	Bus(P2P/P2M)	Yes	Yes

in the Table 2.11, it can be concluded that the characteristics of virtual architectures designed using the method proposed in this Chapter are:

1. Hard cores can be loaded dynamically in slots, with no influence to other running cores. The on-chip communication remains fully active during the partial reconfiguration process and reconfiguration is reliable.
2. Hard cores can be relocated in any slot position. For the 1D model this means that cores can be horizontally reallocated, while for the 2D model approach hard cores can be reallocated horizontally and vertically.
3. Slots can be grouped horizontally (1D and 2D model) and vertically (2D model) to allocate bigger cores without losing relocation capabilities.
4. Slots have access to on-chip specific resources. But, if the resulting architecture is not fully homogeneous, then hard cores that make use of these resources should be allocated in a proper position, next to that resource.

To summarize: virtual architectures, designed following the method presented in this Chapter, have higher flexibility, scalability, make better use of the hardware resources and provide stability during reconfiguration.

More specifically for Xilinx FPGAs, the proposed method permits to overcome the frame based reconfiguration deficiency of Virtex II and Virtex II Pro FPGA at the architecture level and permits to have relocation in 1D and 2D architectures. Also, although the method has been applied for Virtex II based FPGA, it is directly applicable to other FPGAs, for instance Virtex 4 and Virtex 5.

A specific analysis of the on-chip interconnections has been made. Table 2.12 includes a summary of the features of two bus based on-chip communications infrastructures from

the state of the art, described in section 2.1.2, and the two, also bus based, presented in section 2.4.4: the Bus-v1 for an XC2V1000 and the improved Bus-v2 for an XC2V3000 FPGA.

From the data included in Table 2.12, it can be concluded that the Bus-v2 based system has the highest flexibility (relocation possibilities) and integration (more wires in less CLB rows). Compared with the rest solutions, Bus-v2 has more communication wires (Num. Wires = 50) integrated in less area (Used Chip Rows = 4). Also, the frequency and delay results are good (Device freq. and End to end delay in Table 2.12). It is important to mention that the Hübner and Moraes et al. data has been taken from [MHB04b] and [PdMM⁺02] respectively, while for Bus-v1 and Bus-v2, data has been measured with an oscilloscope (frequency) or using the FPGA Editor tool (end to end delay).

Table 2.12: Bus communication infrastructures comparison

	Moraes et al.	Bus-v1 (section 2.4.4)	Hübner et al.	Bus-v2(section 2.4.4)
Device	Virtex E	XC2V1000	XC2V3000	XC2V3000
Num. slots	2	6(real 4)	4	8(real 6)
Slotwidth	NA	4	8	6
Num. Wires	7	12	48	50
Used Chip Rows	4	1 (2.5 %)	6 (9.3 %)	4 (1.5 %)
Data width	1	4	16	16
Used resources	0 Slices 24 TBUFs	32 Slices 20 TBUFs	80 Slices 0 TBUFs	144 Slices 112 TBUFs
End to end delay	NA	NA	5.5 ns	4 ns
Device freq.	NA	60 MHz	66 MHz	75 MHz
Relocation	<i>no</i>	<i>yes</i>	<i>no</i>	<i>yes</i>
Reference	[PdMM ⁺ 02]	NA	[MHB04b]	[KjdITR05]

NA - data Not Available

More in detail, two main factors, listed next, are involved in the achieved higher integration (lower area overhead) of the designed communication structure:

- One of the main advantages of using CLBs as interconnection resources is the possibility of allocating more than one communication structure of this type in one row (advantages have been summarized in Table 2.8). A CLB bus macro with eight wires can start or end in each used CLBs. For instance, if a slot is six CLB columns wide, then up to 48 wires can be allocated in a single row. This property has been widely exploited for building a 2D NoC based reconfigurable system which will be described in Chapter 3.
- The combination of TBUFs and CLB routing resources.

The frequency achieved in Bus-v2 is around 14% higher than the Hübner et al. version, and 25% higher compared to Bus-v1. This is due to the reduced amount of used slices as

route-through macros. In fact, there are no such slices in Bus-v2 and wires are directly routed from the left FPGA corner to the right one. Slices are used only for access points drivers for CLB based wires (TBUF based wires do not need drivers).

2.6 Conclusions

In this Chapter, a method for building flexible, scalable and stable virtual architectures for reconfigurable systems has been originally proposed. Using the method, two bus based systems with one dimensional slot distribution models have been presented and compared with related work, also described in this Chapter. A third, two dimensional NoC based, architecture will be presented in Chapter 3, where on-chip adaptability in reconfigurable systems is the main topic. From the results analysis, it can be concluded that improvements in terms of architecture flexibility and scalability have been achieved, while providing stability during reconfiguration. Furthermore, in terms of on-chip communication, improvements related to the used FPGA rows and frequency have been reported. Also, the method general steps can be applied to other FPGAs.

With the original contributions of this Chapter, the basics for designing flexible reconfigurable systems, that follow the thesis main approach of dividing the hard core design process from the target virtual architecture and the core final location in the system, have been defined.

Reconfigurable Networks on Chip for Reconfigurable Systems



Solutions for on-chip communication in partial RunTime Reconfigurable Systems (pRTRSs). A solution for extending reconfigurability to the on-chip communications is originally proposed.

In this chapter it will be justified that the dynamics of reconfigurable systems require an adaptability extension to the on-chip communications. Several solutions to this problem will be discussed and a thesis contribution in this topic will be presented in detail.

The first part of the Chapter includes a short topic introduction in section 3.1 and related work in section 3.2. Some of the here presented research groups have already been described in Chapter 2, but, here, the focus has been put on the on-chip communications and mainly two dimensional (2D) pRTRSs have been included. Afterwards, in section 3.3, a general analysis of the state of the art can be found. The selected approach and the research work goals can be found in section 3.4.

The second part of the Chapter, starting from section 3.5, presents a ***solution for reconfigurable systems on-chip communication, called Dynamic Reconfigurable NoC (DRNoC), that is an original contribution of this thesis.*** The proposal covers five aspects deeply discussed: i) the architecture definition ii) the supported reconfigurability, iii) the mapping to an FPGA iv) the available design resources and v) the system evaluation parameters.

At the end, in section 3.7, a comparison of the proposed DRNoC architecture and related work can be found. The comparison covers several aspects: i) a structural comparison, ii) a NoC routers comparison and iii) general features summary tables. In section 3.8, the proposed on-chip communication is compared with the bus presented in Chapter 2 and finally, conclusions can be found in section 3.9.

3.1 Introduction

The topic of Networks on Chip (NoCs) is quite new and challenging. It has appeared around 2000 - 2002 in [BM02], [DT01], [GG00] and [KJM⁺] to solve problems related to on-chip communications of highly integrated Systems on-Chip (SoCs), where an arbitrary large number of heterogeneous IP Cores, running at different clock frequencies, have to be interconnected. The communications paradigm adopted is based on globally asynchronous, local synchronous (GALS) communication scheme where IP Cores interconnections are created by circuit or packet switching. In this aspect, concepts from PC networks are translated to embedded system.

In [BM02], the OSI protocol stack has been reassembled for NoCs and called micro-network stack shown in Figure 3.1. The stack is composed of several layers: i) a software layer that corresponds to the system application that includes the application, the presentation and the session OSI layers, ii) the architecture and control layer that includes the transport, network and data link OSI layers and iii) the physical layer that defines the physical connection links. According to [DNAKN], where a different OSI layers grouping is presented and can be seen on Figure 3.1, for NoCs the lower layers (transport, network and data link) are fully implemented in HW (see Figure 3.1).

OSI Protocol Layers	NoC Micro-network stack Bennini & de Micheli	NoC stack Dehyadgari et al.
Application	Software	Software Software/Hardware
Presentation		
Session		
Transport	Architecture and Control	Transaction layer Hardware
Network		
Data Link		Hardware
Physical	Physical	Hardware

Figure 3.1: The OSI protocol stack, the Luca Benini and Giovanni de Michelli stack for NoCs proposal, called micro-network stack and the stack proposed by Dehyadgari et al.

Differently from the PC networks, the communication in NoCs is parallel. Physical links used to connect two routers in the NoC are composed of several wires. The number of wires in NoCs is known phit (physical digit).

NoCs can be based on circuit or packet switching. In circuit switching, a path is formed from source to destination prior to a data transfer by reserving router switches. After data transmissions are finished, router resources are liberated. On the contrary in packet

switching NoCs, that are more common, the path is built while the packet is being transmitted. There are mainly three choices for how packets are forwarded and stored at routers in a packet switching: i) store-and-forward, where a complete packet is stored in a router before taking routing decisions, ii) cut-through, where a packet is forwarded when the header information is available and iii) wormhole, that is the most popular and studies switching methods where a packet is split into several flits (flow control digits). In wormhole switching the routing is done as soon as possible, similarly to cut-through, but here only one flit is needed to be received by the router in order to forward it to the next one. As a result, the packet may be spread into many consecutive routers, like a worm. Usually, in NoCs, flits (flow control bits) and phits (physical digits) have the same width. The most common width in NoC based ASICs is 16, 32 or 64 bits, but there are some examples where up to 256 bits are assumed [SKH08].

A few years after the first appearance of NoCs, the community working in reconfigurable and adaptable systems started studying the use of NoCs for on-chip communication. In such dynamic systems, where the final application is not known at design time and it can change at runtime, the selection and design of optimal communication strategies that cover "unknown" requirements is a great challenge that results in the need of flexible and adaptable on-chip communications.

A direct and general solution for this problem is to have an over-dimensioned and fixed on-chip communication infrastructure, where routers and communication logic in general are also fixed in the reconfigurable device or even melt in the silicon. An example of a general purpose NOSTRUM NoC instance is a 128 (256 wires) bit mesh NoC [MNT⁺04]. On the other hand, ad-hoc NoCs are preferred when a specific application or a set of applications are targeted. For instance, the design flow presented in [SCK07] targets to minimize the number of used network routers and NoC links. Additionally, it is a well known fact that NoCs can be considered an option when more cores have to be interconnected and some authors say that the threshold number is 10 cores. Therefore hierarchical NoCs have appeared, like HiNoC presented in [HLZ⁺06] and [HG07]. Hierarchical NoCs are solutions where buses are used for local connection and NoCs for global connections. These NoCs attenuate problems related to NoC high area overheads and latency, when the number of locally connected cores is low.

On-chip communications adaptability in commercial, fine grain, reconfigurable systems (FPGAs) can be achieved by two general approaches:

1. Online routing. To route the needed cores interconnections right before loading them into the FPGA. This option is not suitable for runtime reconfigurable systems as it is conceived today. Routing individual CLBs and BRAM Switch Matrixes (SMs) at a low granularity level is unaffordable. More specifically, each SM in Xilinx Virtex II FPGAs has around 400-500 input/outputs and the routing information in an FPGA bitstream is around 70-80 % of all the configuration data, thus the complexity of the routing is extremely high for executing it at runtime. There are several works that are oriented to solve this problem and provide faster solution, like [KPG07], [GASF03] and [SF06b]. In the last one, which is more related to FPGAs, a solution where several hard cores are linked, on a host station, prior to be loaded in a system is presented.
2. Routing by interconnection reconfiguration. It consists on modifying the on-

chip communication by exploiting partial reconfiguration. This, higher lever option, deals with coarse granularity (for instance change switch matrices that interconnect IP cores), therefore it is more restrictive, but less time consuming and can be executed directly on an embedded system. This option is the one followed in some related work groups (for instance, CoNoChi and Hübner et al.) and also in this thesis.

In the next subsections, several academic NoC solutions for runtime reconfigurable systems are described in detail. It is important to remark that differently from other Chapters related work, this one does not include commercial solutions. Although SoC design companies, like ST Microelectronics and Philips, already have their own NoCs for ASIC design, until now, none FPGA provider includes packet switched NoCs in its reconfigurable systems designs. This section is also used to introduce some concepts needed within this Chapter. However, before that, simply to give an idea of the evolution of the research topic, a time graph of the groups included in this state of the art can be seen in Figure 3.2. From the Figure it can be noticed that most of the solutions included in the state of the art section, have firstly appeared in the same year (2006).

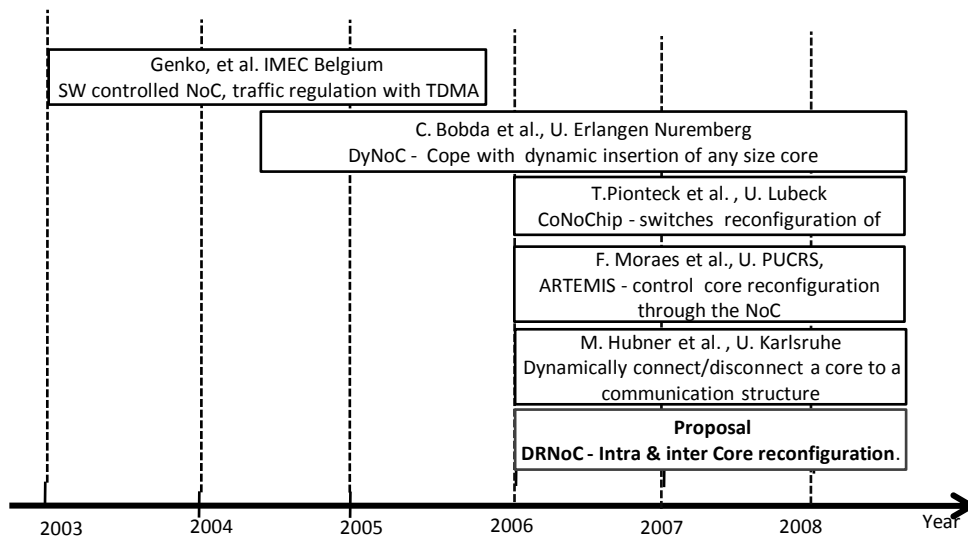


Figure 3.2: NoCs for reconfigurable systems research topic - time evolution. The graphic includes all the research groups described in this Chapter, along with the proposed solution.

3.2 Academic Related Work

The on-chip reconfigurable communication solutions, presented in this section, have been compared with the thesis proposal in section 3.7.1, using metrics defined by Pionteck et al. in [TPB07] and complemented with other defined in this Chapter. All new and former metrics definitions can be found in section 3.6. The main characteristics of each related work to be highlighted are: i) system architecture ii) NoC characteristics, iii) use cases and iv) practical implementation.

3.2.1 Nollet-Mignollet et al.

One of the first groups working on the use of NoCs in dynamic reconfigurable systems was IMEC. In [NCV⁺03], [MNC⁺03], [NMV⁺04] and [MNM⁺04], the future reconfigurable system on-chip that was foreseen was a mesh of heterogeneous components (tiles), some of them reconfigurable, interconnected by a NoC. A general view of the proposed architecture can be seen in Figure 3.3. The work is focused on the software control of IP cores loaded in reconfigurable tiles for on-chip data traffic regulation. For achieving this, Nollet-Mignollet et al. use three types of NoCs in a single reconfigurable system: a data network that transmits application data, a control network that sends control data and a reconfigurable network, used to transmit new configuration data to reconfigurable tiles.

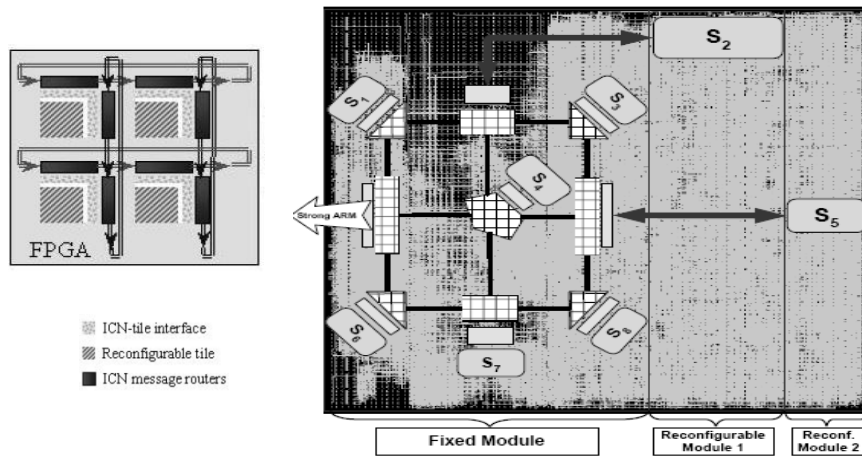


Figure 3.3: Nollet-Mignollet et al. proposal. Architecture general view on the left. Virtex II FPGA implementation on the right [NMV⁺04].

NoC adaptability is foreseen in two aspects: i) change routing tables content, where new configurations are transmitted through the control network and ii) regulate the amount of time an IP core can use the NoC by Time Division Multiple Access (TDMA). This period of time is regulated by sending commands to the NoC Network Interfaces (NIs) using the control network.

The network use by Nollet-Mignollet et al. has the characteristics listed next:

1. Topology - Can be defined at design time.
2. Routing - Routing tables.
3. Buffers - FIFOs at the outputs.
4. Arbiter - Not Specified.
5. Switching - Virtual Cut-through

The network design resources, used by Nollet-Mignollet et al., provide the possibility of defining the NoC topology at design time and also, to independently select the number of a router port inputs and outputs, allowing to have unequal number of inputs and outputs. Router FIFOs are allocated at the outputs, after the routing process. This approach forces FIFOs content to be reorganized or to be emptied if a core connected to the NoC changes its position or a new one is loaded.

The Nollet-Mignollet et al. solution for mapping different applications to reconfigurable tiles is interesting. If, for instance, a task has to be mapped to a microprocessor tile, but this tile cannot provide the required communication resources, because it is too far from the other ones involved in the current application, the adopted solution is to load a soft microprocessor core in the nearest reconfigurable tile and execute the task on it. The proposed network has been implemented on a Virtex II FPGA using a 1D architecture model for the reconfigurable system as it can be seen in Figure 3.3, where reconfigurable modules allocated on the FPGA right side (S2 and S6 on the figure) occupy the entire FPGA high and are commented individually with horizontal macros to a NoC router. As a real application, a 3x3 with two reconfigurable tiles has been implemented and can be seen in the same Figure. Each reconfigurable tile was composed of FPGA LUTs, BRAMs and multipliers. The remaining, seven, tiles as well as the entire NoC were part of the FPGA fixed area and could not be reconfigured.

3.2.2 Hübner-Becker et al.

The bus based communication system designed in Karlsruhe group, deeply described in Chapter 2, was adapted to permit ring NoC connections in [MHB04a]. For this aim, accesses to the ring NoC have been divided into time slots, and a master-module provides a five bit counter to each slot that defines the time a slot can access the bus lines.

After that, a flexible multiprocessor system on-chip where each node of the NoC mesh was composed of a soft-core processors (MicroBlaze), a local memory and a timer, was presented in [MHB05]. The used reconfiguration approach in that case is to load new software programs in the corresponding processor, instead of changing the FPGA configuration. Differently, in the latest publications, [MHB], [MHB06], [MHB], [JBP06], [JBH] and [MHB08], a 2D reconfigurable system is presented. A general view of the system architecture can be seen on Figure 3.4. The approach followed in [MHB] is based on the idea of a simplified and localized on-chip routing adaptation.

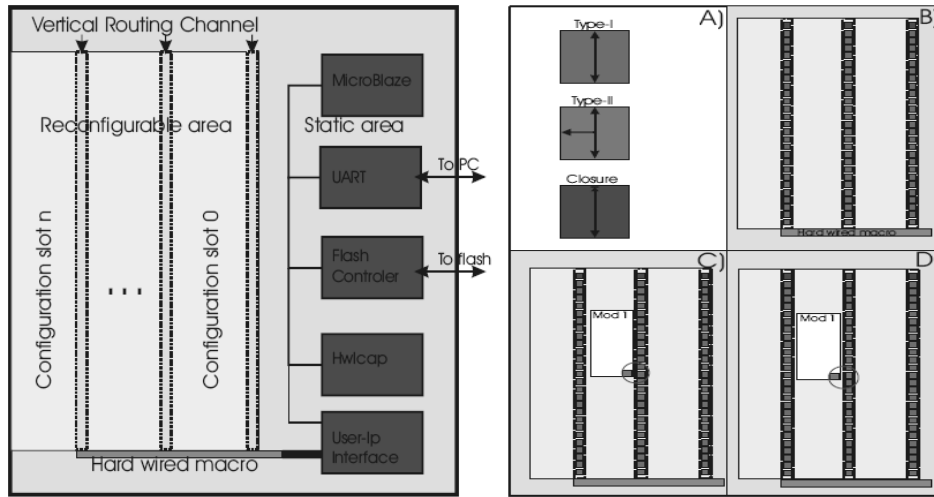


Figure 3.4: Hübner-Becker et al. 2D system, focused on-chip connections adaptation based on partial reconfiguration. The system permits a core to be connected/disconnected to a communication structure. On the left, the architecture general view and on the right, the connection reconfiguration resources [JBH].

The basic element of reconfigurable system is a reconfigurable module that is composed of a routing module and a function module. Reconfigurable modules can be allocated in any horizontal slot or vertical position within a slot, which is the main advantage of the proposed system architecture. For this aim, horizontal LUT macro structures are placed along one side of each defined slot. The vertical distance between each horizontal LUT macro defines the step for the vertical placement of a reconfigurable module. The resulting areas are called cells. This cell based reconfigurable system main advantage is that it permits to place multiple small modules that do not use entire slots in the same slot that results in good use of the available FPGA resources. For making the allocation of multiple reconfigurable modules in one slot possible, vertical routing channels are defined next to each slot (see the left side of Figure 3.4). All vertical routing channels are connected to a horizontal hard wire macro allocated in the bottom part of the FPGA that extends the system fixed area.

Routing is based on two types of communication primitives (see Figure 3.4, right side), Type I that is a vertical feed-through and Type II that apart from the vertical connection has an additional branch that connects signals horizontally to the left. To load a new reconfigurable module in the system, first it is properly placement within a slot and then, it is connected to the hard wired macro on the bottom of the FPGA using an appropriate combination of Type I and Type II primitives that are loaded into the vertical routing channel. If a reconfigurable module has to be disconnected from the system then, the corresponding Type II primitive is replaced by a Type I. This simplified routing procedure makes possible to connect/disconnect reconfigurable modules to/from the system.

The presented implementation done on a Virtex II Pro FPGA is composed of 8 cells

distributed in single reconfigurable slot. Each cell is composed of 220 CLBs, among which the routing modules use 62 CLBs and the required connections 28 CLBs. The remaining 120 CLBs can be used for the hard core functionality [MHB].

The implemented routing modules contain routers with 8 bits width connections. For making the routing in the system possible, a unique ID, coded from vertical and horizontal coordinates is assigned to each slot cell. The routing algorithm implemented by the routing modules calculates the distance in both dimensions and selects an output port.

Regarding the software support for partial reconfiguration, the approach followed is to read back the FPGA configuration, then to merge the new reconfigurable module, route it (assigning corresponding primitives) and finally to configure the FPGA. This technique is known in the literature as Read-Modified-Writeback. In the first versions, the Read-Modified-Writeback was based on JBits and ran in a host PC, but in the latest publication [MHB08], it seems that all the system has been ported to a MicroBlaze soft-core processor. In the future, the group wants to improve the system and targets real online routing without the use of primitives.

3.2.3 Bobda et al.-DyNoC

Christophe Bobda et al. from the University of Erlangen-Nürnberg, first addressed the problem of interconnecting constantly changing modules in reconfigurable systems based on NoCs in [BMK⁺04], [CBvdV05], [BA05] and [BA05]. The solution proposed is a NoC called "DyNoC - Dynamic Network on Chip" and is shown on Figure 3.5. Bobda et al. define a set of requirements to the communication architecture to support dynamic changes and to achieve better logic/programming elements (PE) ratio, listed next:

1. PEs should be flexible, coarse grain, elements because otherwise the network wrapper logic/PE becomes big and a lot of area is wasted.
2. Each PE should have access to the NoC.
3. PEs should directly communicate with their neighbors
4. The network logic has to be flexible enough to be used within the module to which it belongs.

A general view of the communication infrastructure proposed by Bobda et al. is a mesh of interconnected routers, and it can be seen in Figure 3.5 - left side. The main problem addressed by Bobda is the loading of different sized cores and to achieve a routing strategy that could cope with this problem. The problem statement is as follows: in a static NoC, a router has always four active neighboring routers, but when dealing with the DyNoC architecture, the area occupied by a router can be covered by a module during reconfiguration. Since those routers cannot be used, they are deactivated. When the module execution has finished and the module area becomes free, routers are activated again. From this concept, it can be deduced that in the DyNoC approach, routers are fixed elements in the architecture.

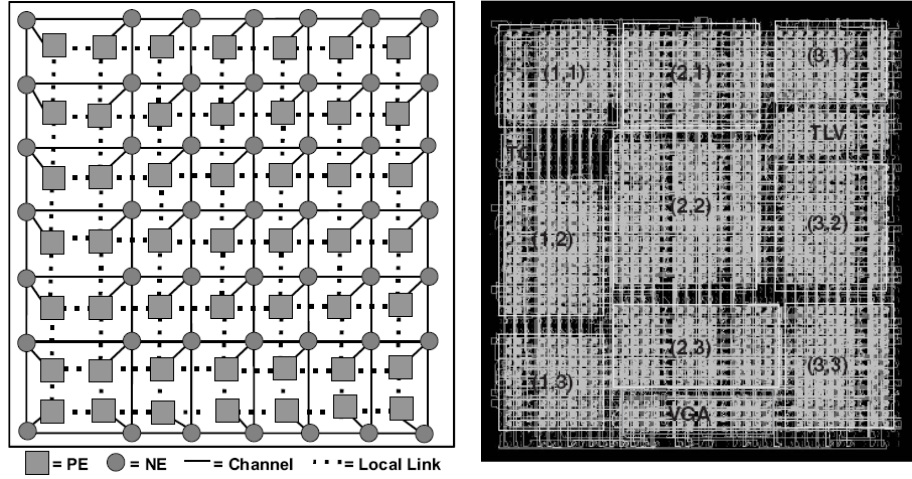


Figure 3.5: The DyNoC architecture, proposed by Christophe Bobda et al. from the University of Erlangen-Nürnberg. It permits to load cores with different size. The architecture general view can be seen on the left size and on the right side the implementation on a Virtex II FPGA [BA05].

For achieving flexibility, Bobda et al. have developed a modified XY surrounding routing algorithm [MMT05]. According to Bobda et al., in the modified version of the XY algorithm, routing decisions are taken locally and therefore there is no need of routing tables and this permits routers to occupy much less area. Routers with XY routing can operate in three modes:

1. N-XY mode - Normal XY mode. Routers behave as normal XY routers where data is first router un X direction and then in Y direction.
2. SV-XY mode - Surrounding mode when a vertical (left or right) neighbor has been deactivated.
3. SH-XY mode - Surrounding mode when a horizontal (up or down) neighbor has been deactivated.

Another, interesting, option is to use an adapted Q routing algorithm which is an adaptive learning routing algorithm. This type of routing is distributed and aims to minimize the delivery time. Each router runs its own Q routing algorithm using only local information from its direct neighbors.

A summary of the DyNoC characteristics is listed next:

1. Topology - Mesh
2. Routing - Surrounding XY and Q routing
3. Buffers - FIFOs at the input.
4. Arbiter - Round robin.

5. Switching - Wormhole

The use case, reported by Bobda et al., is a light controller that targets the testing of the surrounding routing algorithm in a 3x3 DyNoC mapped on a Virtex II FPGA (XC2V1000), see Figure 3.5 - right side, where three slices are connected by macro structures to the NoC and one of them is partially reconfigurable.

3.2.4 Pionteck et al. - CoNoChi

Pionteck et al. from the University of Lübeck - Germany presented in [TPG04] and [TPG] a reconfigurable Network processor, DynaCOREs, that can be adapted to different traffic loads. Depending on the network traffic, a new hardware assists was loaded into the FPGA. Hardware assistants allocated in the FPGA are connected by a NoC. Afterwards in [TPA06], [TPK06] and [TPB07] the need of adaptability in the NoC infrastructure has been addressed, and a solution has been proposed. The solution is called "Configurable Network on Chip - CoNoChi". The NoC consists of switches with four full-duplex equal links. The links can be used either for connections to adjacent switches or for connecting a hardware module to a switch (see Figure 3.6). Depending on the number of non local links, a linear network or a 2D grid can be instantiated. In this way, CoNoChi provides a dynamic communication infrastructure for reconfigurable modules that permits to allocate cores of different sizes by dynamically inserting and removing NoC switches using partial reconfiguration.

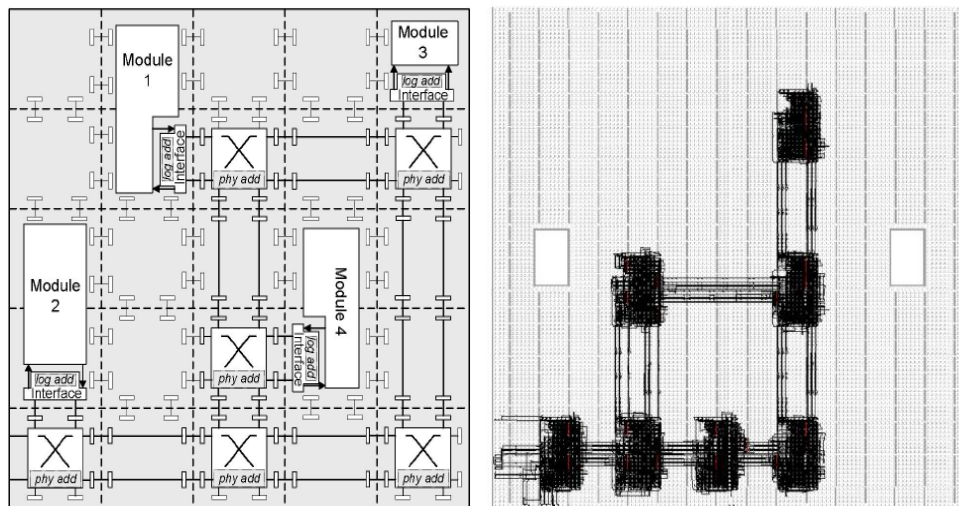


Figure 3.6: Pionteck et al. on-chip communication solution, called CoNoChi, general view on the right side and an implementation on a Virtex II Pro on the left side. The architecture permitted reconfigurability is to load different size cores and to add new routers to the NoC.

Pionteck et al. also focus on problems related to the routing policy during the adaptation process. The presented solution is based on routing table updates by sending the

new tables contents and control data using the same NoC. The method used is as follows: i) first data in the network are re-routed to free the connection link where the new router will be included. Thus permits data in the NoC to be transmitted using switches not involved in the reconfiguration, then ii) switches related to the link under reconfiguration are disabled to avoid glitches. After partial reconfiguration, iii) affected routers routing tables are updated through the network.

The CoNoChi NoC and switches have the characteristic listed next:

1. Topology - MESH.
2. Routing - Table routing.
3. Buffers- FIFOs implemented in BRAMs at switch inputs. The size of the FIFO is one NoC packet.
4. Arbiter - Priority combined with roundrobin.
5. Switching - Wormhole

A general view of the CoNoChi architecture mapped to an FPGA can be seen in Figure 3.6 on the left side, while on the right side the implementation of the approach on a Virtex II Pro FPGA can be seen. The FPGA is partitioned into a grid of rectangular modules that can be configured as a switch, as a horizontal or a vertical communication line or as part of the hardware module. Configuration data for the different variants is stored in the local memory. The presented mechanism implies that all the unused areas of the grid are configured to connect at least default communication lines and LUTs. In the presented use case, a big hard core or module is substitute by two smaller ones and a switch for one of the new modules is added.

Runtime SW support has not been specified, neither for bitstreams manipulation, nor for architecture management. But it has been reported that the algorithm used for routing table generation can be ported to the Virtex II Pro PPC.

Analyzing the presented by Pionteck et al. approach, and taking in mind that in the general approach the relative location and the used resources by switches and cores is not restricted, it can be noticed that the provided flexibility is very high. However, this results in that switches links length freely changes along with the link delay, and the control system has to cope with this aspect during the links building process. In the solution proposed in this thesis, the same problem of increased link delay appears, but since the routers possible positions are restricted, the problem is attenuated.

3.2.5 Moraes-Calazans et al.

The Fernando Moraes, Ney Calazans et al. group from Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre Brasil, have designed a NoC called "HERMES" [MCM⁺04b], [MCM⁺04a].

The main characteristics of the Network are listed next:

1. Topology - MESH routers can have up to five ports.

2. Buffers - Input FIFOs.
3. Routing - XY,
4. Arbiter - Roundrobin.
5. Switching - Wormhole

More recently, the same group has presented a network called "ARTEMIS", based on the HERMES NoC, in [MGC⁺07] [MMCM05] and [MGCM06]. Differently from the HERMES NoC, ARTEMIS includes the possibility of dynamically reconfiguring cores connected to it. For this aim, several LUT based macro structure that connect neighboring modules have been designed and used in the Network Interface (NI) to NoC border. Differently from other macro structures, like the ones presented in this thesis, the presented in [MGCM06] include control lines used to enable and disable the data transmission through the macro that, in some cases, could be advantageous. For controlling core reconfiguration, control packets are sent through the NoC. These packets can connect or disconnect a core to a router.

As a use case a 2x2 NoC with two reconfigurable cores has been presented mapped in a Virtex II FPGA using a 1D architecture model. The selected use case application is a reconfigurable ALU that can switch between a divider, a multiplier and a square root. The entire system can be seen in left side of Figure 3.7, while the floorplanning of the presented use case can be seen on the right side.

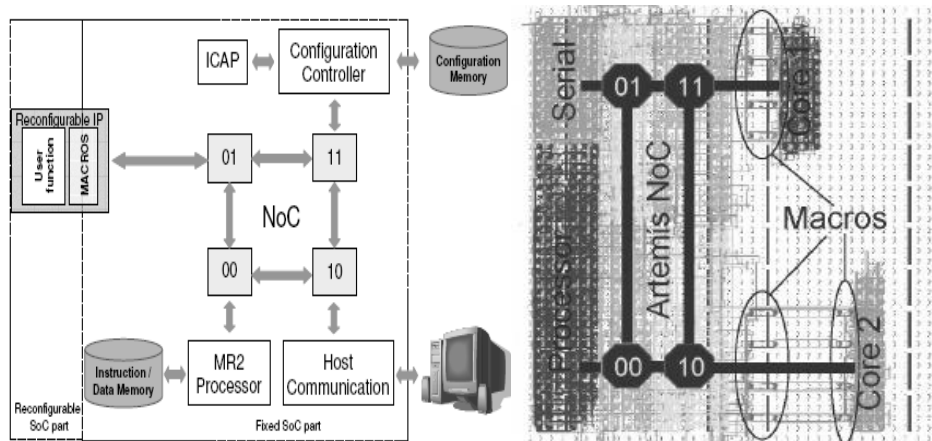


Figure 3.7: Moraes-Calazans et al. ARTEMIS NoC. A general view on the left and a Virtex II implementation on the right side. This NoC permits nodes to be reconfigured.

The main advantage of the HERMES network is its small size and its modular design. This, as well as the open source approach of the research group, have lead to the selection of this NoC for the purpose of this thesis. Some details of the NoC will be discussed later in section 3.5.3, when the modifications applied to adapt it to fit the NoC 2D reconfigurable system presented in this Chapter will be remarked.

3.3 Problem Statement

As it has been mentioned, one of the main aspects in reconfigurable systems is that the final application is "unknown" in advance and it may change during device utilization. Even more, each core to be loaded in the system has its own communication requirements that have to be granted by the underlying communication architecture. From the state of art section it has been clarified that flexibility, adaptability and scalability are needed in the on-chip communications to meet this challenge.

The current approaches try to bring this flexibility, adaptability and scalability starting from the assumption that the on-chip communication is a homogeneous NoC (all communication elements are the same type and/or have the same architecture). If a new core is loaded/removed, then it will be connected/disconnected to the same NoC. Here, an exception has to be mentioned: in [TPB07], Pionteck et al. mention that custom modules can be added to the system, but does not give any further specifications and therefore this will not be considered.

3.4 Selected Approach and Goals

Differently from the state of the art, *the approach followed in this thesis is to define a general on-chip communication infrastructure, that is then customized for a given application. The original proposal, which is called Dynamic Reconfigurable NoC (DRNoC), provides the possibility of building ad-hoc communication strategies (NoC, bus, P2P, P2M hierarchical or a mix) for a given application, on one hand, and, on the other, to different application, with their own communications, to coexists in the system.*

More in detail, the goals to be covered by the DRNoC virtual architecture communication infrastructure are listed next:

1. *To provide the possibility of defining fully independent communication schemes for each application that is loaded in the system.* This characteristic will be refereed as independent communication.
2. *To provide the possibility of building communication schemes of any type including heterogeneous (mix of P2P, P2M and/or NoC).*
3. *To provide the possibility of extending an already defined communication scheme.*
4. *The solution has to be scalable, flexible and to permit an optimal use of the on-chip resources.*
5. *To provide the adaptation capabilities to the lower layers of the NoC protocol stack.* The goal here is to bring reconfigurability to all those layers fully implemented in HW (see Figure 3.1 in the introduction section): i) the transport and the network layers that provide the end-to-end connectivity, ii) the data link layer that is the responsible for the reliability of the path creation across links and iii) the physical layer.

3.5 Dynamic Reconfigurable NoC - DRNoC

A general view of the Dynamic Reconfigurable NoC (DRNoC) architecture for runtime reconfigurable systems, proposed in this thesis, can be seen in Figure 3.8.

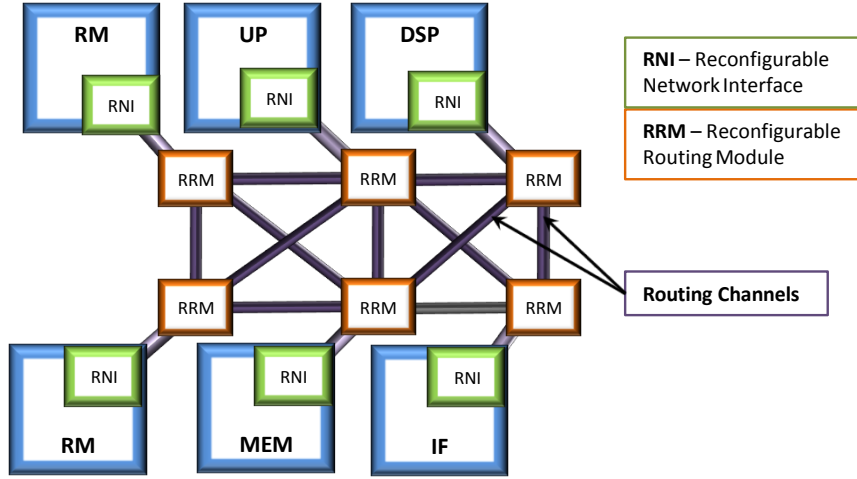


Figure 3.8: DRNoC Architecture General View. Heterogeneous cores are interconnected by an Xmesh of reconfigurable routing modules. Cores access the communication network through reconfigurable network interfaces.

3.5.1 Dynamic Reconfigurable NoC-Definition

The Dynamic Reconfigurable NoC (DRNoC) is composed of a set of heterogeneous elements connected through *Reconfigurable Network Interfaces (RNIs)* to an on-chip mesh of *Reconfigurable Routing Modules (RRMs)*. Each element will be defined further in this section. The mesh also includes diagonal links that will be referred as Xmesh. These diagonal links are not usual in NoC design, and of course in a silicon die diagonals cannot be made and these connections are indeed "S"-like. Diagonal links have been included in this solution for achieving more flexibility, to cover the set requirement of independent communications and to give support for different types of communication schemes.

For instance, Figure 3.9 shows a 6x6 Xmesh with four fully different communication schemes mapped on it: i) a regular mesh NoC that uses two diagonals on the top-middle part, ii) an irregular mesh NoC on the left, with a "large core" allocated in the top, iii) one P2P in the bottom and iv) one heterogeneous hierarchical communication scheme on the right, where a P2P connection is used locally and a star NoC is used globally. A floorplanning like this, where all nodes are distributed in fully independent communication schemes, would not be possible if the diagonal links were not available. An alternative to the Xmesh diagonal links is to concatenate several links in an "L"-like connection, but in that case latency will not be equal in all links, and resources from other links will be required.

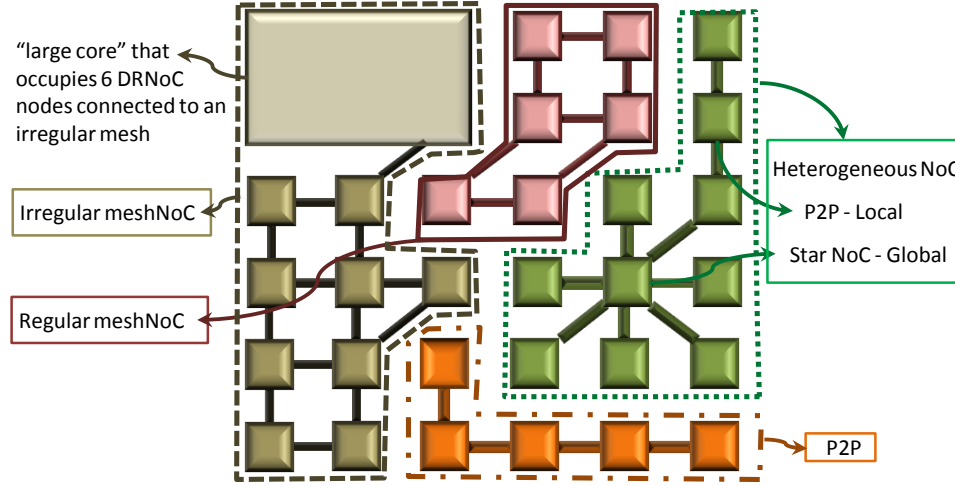


Figure 3.9: A 6x6 DRNoC architecture with four independent communication schemes mapped.

The main elements of the reconfigurable communication are listed next and some DRNoC characteristic are described after that:

1. **Reconfigurable Network Interfaces (RNIs)** define a standardized interface between cores and the network. They are in charge of an appropriate data formatting. The main Network Interface (NI) configurations that can be loaded in RNIs are: serializers, deserializers, packetizers, depacketizers, multiplexers, demultiplexers, buffers, feed-throughs and empty designs. All these elements provide reconfigurability at the transport and network NoC layers.
2. **Reconfigurable Routing Modules (RRMs)** define the application communication strategy. Routing modules can be: buffers, any type of NoC router with up to eight ports that fit in the reserved RRM area, switches, multiplexers, feed-throughs and empty designs. Depending of which aspect is modified, these elements provide reconfigurability at the transport and network layers and/or the data link layer. For instance, according to [DNAKN], virtual channels are part of the transport and network layers, while the switches communication protocol belongs to the data link layer.
3. **Communication Channels** define the communication physical links. They are composed of a fixed number of wires with specific characteristics and position. To reduce problems related to link delays, these are the only elements that cannot be changed in the system (like silicon metal wires). If a larger distance connection (no neighboring nodes) is required, then buffers have to be added in RRM along the path. Communication channels are divided into subchannels that are composed of an equal amount of wires. All channels but the local one, the one that connects RNIs with RRM, are equal. The local channel size defines the amount of independent communication schemes a core can take part in. Since each RRM is

commented through communication channels to all its eight neighbors, the ideal size of a local channel is equal to $8 * channelswires$.

4. **Switch Matrixes (SM).** These elements are indeed part of RRM, but are listed here separately because they may be reconfigured separately if the architecture mapping permits it. SMs define which communication channels will be used and thus provides reconfigurability to the physical layer in the sense that the overall communication scheme topology can be modified.

Each RRM has eight communication channels used to build the Xmesh. Each element of the DRNoC communication infrastructure has a fixed, localized position(s) in the system, and is composed of specific FPGA resources. In the current implementation all of them are composed of CLBs, but in other FPGAs, like the Virtex 4 or Virtex 5, where on-chip FIFOs are available, these specific components might be part of reconfigurable routing modules because FIFOs are a mail element in routers.

The RRM area and the available channels limit the amount of independent communication elements that can be loaded in it, and thus, the number of independent communication schemes that can coexist in the system. Differently, the RNI area and the available wires in the local channel define the amount of communications a core can take part of (for instance in heterogeneous NoCs, the bus to NoC bridge takes part in two communications schemes). Every communication scheme mapped into a DRNoC can use any amount of the available channels (1, 2, ..) and an integer amount of subchannels.

The group of an IP core, a RNI, a RRM and a switch matrix configuration will be refer as a **DRNoC Reconfigurable Node (RN)** or simply as DRNoC node. Some aspects of the presented approach are similar to others. In DyNoC, for instance routers have also a fixed position and similarly to CoNoChi, a switch can be changed by a feed-through. But, DRNoC intends to go further by providing the possibility of defining individual and, ad-hoc, communication strategies (not restricted only to NoCs) and bring reconfigurability to the NoC protocol stack lower layers.

The DRNoC on-chip communication solution will be compared to other approaches in section 3.7.1 using a set of defined structural parameters in section 3.6.2.

3.5.2 DRNoC Mapping to a Xilinx Virtex II Virtual Architecture

Two of the 2D virtual architectures, described in Chapter 2, have been used for mapping the DRNoC approach into two different FPGAs, an XC2V3000, which is the physically available and on an XC2V8000, just to test the scalability of the architecture. A general view of a 2D virtual architecture can be seen in left half of Figure 3.10. A detailed description of the architecture can be found in Chapter 2, and the needed adaptations for mapping the DRNoC approach can be found next.

Each DRNoC reconfigurable node (RN) is composed of four elements, RRM, RNIs, SMs and the IP cores represented by FPGA hard cores. For achieving the highest flexibility and adaptability, the best mapping of the DRNoC architecture will be to have one reconfigurable slot for each DRNoC reconfigurable node element. Thus, three or four slots will be needed (depending if an entire slot will be assigned to SMs, or they will share one slot with RRM). Here, a mapping problem appears, which will be discussed

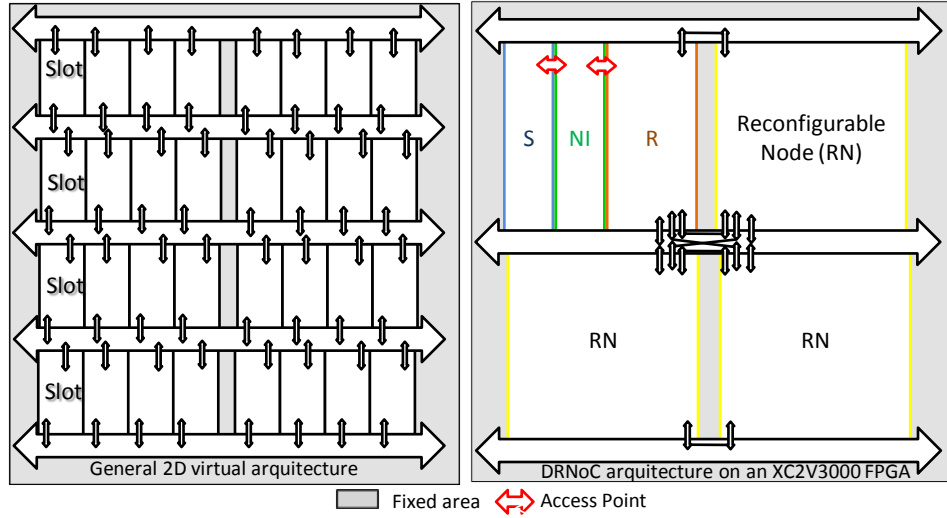


Figure 3.10: On the left side, a general view of a 2D virtual architecture designed with the method defined in Chapter 2 and, on the left side, the DRNoC approach mapped on a slightly modified virtual architecture.

using an XC2V3000 FPGA virtual architecture from Chapter 2 as example. If the three slots option is selected, the regularity of the system will be disturbed as there will be always one slot left from each DRNoC node mapping (see Figure 3.10 left side) and this slot will have to be assigned to the fixed area, while in the four slot SMs have to be allocated in an entire slot that will result in wasted more area.

Looking at Table 6 in Chapter 2 (Virtex II 2D slot distribution), there are two suitable slots distribution (for the XC2V3000) to choose between: first, a 2x4 with 12 CLB columns wide slots (slot size of 288 CLBs) and second, a 2x8 with 6 CLB columns wide slots (slot size of 144 CLBs). Estimating that the area needed for state of the art routers with three inputs is around 150 CLBs, and that part of the available CLBs will be used for the communication channels implementation (bus macros), the best slot size, with respect to RRM, is one higher than 200 CLBs. which corresponds to the first slot distribution (2x4). But, on the other hand, going back to the mapping problem, if the first VA (2x4) is chosen the mapping of either the 3 slots or the 4 slot option will result in 2x1 DRNoC that is a very restrictive mapping for test.

The final decision is as follows: first, the slot division with smaller granularity will be used, the 2x8 VA (with 144 CLBs slots), and second, RRM and SMs will share the same double sized slot (288 CLBs) and, if there is any problem due to Xilinx place and router tools, slots can further be grouped. This results in a 2x2 DRNoC implementation, on the XC2V3000 FPGA, with two slots assignment to cores and NIs respectively, and one double size slot to RRM and SMs. In the right side of Figure 3.10, the DRNoC mapping to the 2x8 VA slots is shown and, from here after, the RRM and SMs slots will be marked with *R*, RNI with *NI* and cores will remain with an *S*. Cores to be mapped in *S* will be referred as slot/core to be differentiated from the general term "slot".

With this, the slots distribution and DRNoC node elements mapping has been defined and now, in the next paragraphs, all DRNoC connection links will be assigned to the general virtual architecture macro connections, and the needed adaptations will be explained.

Cores communicate only with network interfaces, that is S needs to be commented to NI (see Figure 3.10 right side), while network interfaces communicate with routers, that is NIs need direct connection to RRM (see Figure 3.10 right side). Thus, neither S, nor NIs need direct access to main horizontal communication channels (see Figure 3.10 right side). Therefore, these connections have been replaced by horizontal neighboring connections, implemented with six CLBs wide BM-LUTs bus macros (this type of macro has been explained in Chapter 2).

Regarding the Xmesh communication channels implementation, they have been entirely created with bidirectional vertical BM-LUTs that pass four wires from a RRM to the main horizontal communication channel, and four wires from the horizontal communication channel to RRM.

Communication channels side BM-LUTs input and outputs are connected in the Xmesh and routed with the Xilinx ISE PAR tool using only the area reserved for the channel. This approach, that is different from the one used to design the 1D bus based system presented in Chapter 2 (where the communication infrastructure was designed manually with the FPGA Editor tool), permits to reduce the communication infrastructure design time.

Apart from all the described connections, there are some wires individually routed to each slot, use to pass control signals (run, reset and reconfigure) to each node element, like in the 1D system. These signals are managed by the control system and also, additionally, a special bus used for performance measurements data extraction, from cores and network interfaces, has been added to every DRNoC row. The bus and the control signals are allocated in the rows reserved for the 2D communication channels. Reconfigurable routing modules are not connected to the measuring systems, in the current implementation, in order to save some area. The defined RRM double sized slot has 288 CLBs, and that is not enough for mapping together, routers, SMs and the measuring system registers along with all the needed communication macros.

However, the needed and defined modifications of the original 2D virtual architecture to map DRNoC, resulting in a DRNoC VA, have not affected the original architecture properties. As any other virtual architecture, designed following the method proposed in Chapter 2, the DRNoC virtual architecture permits: i) cores to be allocated in any slot/cores position, NIs in any RNI position and routing elements in any RRM position ii) DRNoC nodes can be grouped to allocate bigger cores (cores that need to use more than a DRNoC node) and iii) the on-chip communication is not affected during partial reconfiguration.

Two example DRNoC implementations for an XC2V3000 and an XC2V8000 FPGAs have been created and can be seen on Figure 3.11. For both FPGAs, the defined DRNoC connectivity has the same implementation characteristics which are listed next:

1. Communication channel width is 80 wires, 40 per direction, divided into four subchannels of 10 wires each.

2. Local channels, between RRM and RNIs, are composed of two channels with a total of 160 wires, 80 per direction.
3. Access points, between slot/cores and RNIs, have 152 wires, 76 per direction. This number for instance is enough to implement a full AMBA APB master-slave duplex communication.
4. Each slot/core, RNI and RRM has four control wires.

The previous list does not include the measuring system connectivity (AMBA APB bus) as it is not considered part of the DRNoC architecture.

A view of an empty (all slots free) 2x2 DRNoC VA for the available in the currently used development board XC2V3000 FPGA can be seen in Figure 3.11 on the right side and, for demonstrating the scalability of the proposed DRNoC architecture, a 4x4 DRNoC VA for an XC2V8000 FPGA, the biggest of the Virtex II family, is shown in Figure 3.11 on the left side.

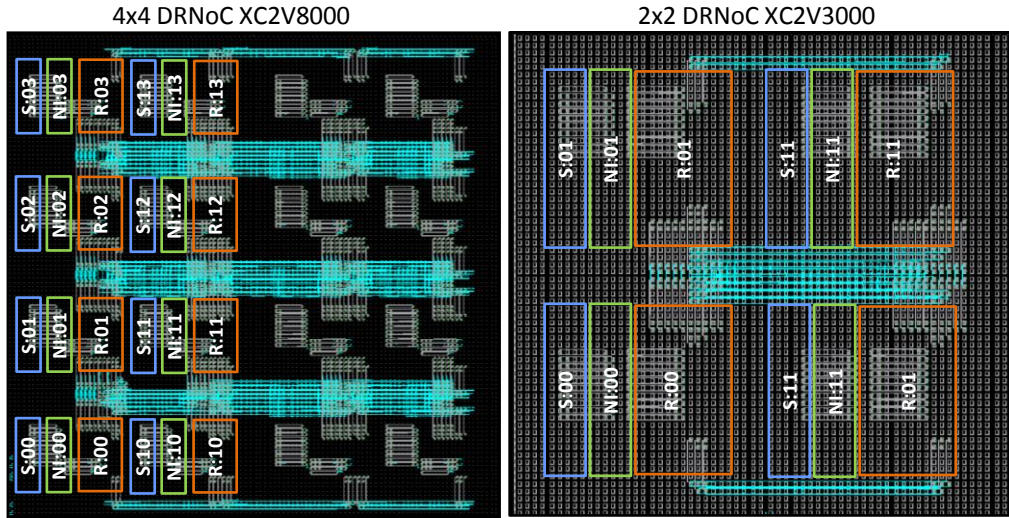


Figure 3.11: DRNoC mapping to two Virtex II FPGAs: a 4x4 DRNoC on an XC2V8000 on the left and a 2x2 DRNoC on an XC2V3000 on the right.

It is important to know the real area that is available for logic implementation after placing this huge connectivity in the FPGA. As it has already been mentioned, the internal connectivity has been entirely designed with BM-LUTs, and every BM-LUT uses 1 CLB (4 slices) in each side involved in the communication. For instance, for the 152 wires of the slot/core to RNI access point implementation, 19 CLBs are needed from each side. A summary of all the required resources for each DRNoC architecture for both, XC2V3000 and XC2V8000 can be found in Table 3.1. Worst cases for RRM have been considered for both FPGAs: for the 2x2 XC2V3000 implementation, the worst case is a RRM with three channels, while for the 4x4 XC2V8000 it is with eight channels (see Figure 3.11). The Table includes the percentage of the needed CLBs from each slot

assigned to a DRNoC node element, the total amount of the needed BM-LUTs in each DRNoC node and the total percentage of CLB resources needed for the entire DRNoC connectivity.

Table 3.1: Two example DRNoC implementations on Xilinx Virtex II FPGAs VA.

	Used slot/core CLBs (%)	Used RNI CLBs (%)	Used RRM CLBs (%)	Real slot/core Size (CLBs)	Real RNI Size (CLBs)	Real RRM Size (CLBs)	DRNoC node No. BM-LUTs	Total FPGA (%)
XC2V3000	13.1%	27%	17.3 %	125	105	238	108	12 %
XC2V8000	15.8%	32.5%	41,6 %	101	81	120	158	21,7%

For the purpose of this thesis, the most important values from the Table are the real slot/core, real RNI and real RRM sizes as these values define the hard cores that can be loaded on them and the limit of the system scalability. These values are in the range of 100 CLBs for cores and NIs to be loaded in slot/core and RNIs and 200 CLBs for routers to be loaded in RRM. Having into account the problems related to the current Xilinx PAR tools versions (it is not possible to restrict the routing inside a defined area), the available area is quite low and this will restrict the possible systems that can be loaded on the pRTRS. The experience gained has shown that for having a module fully routed inside the defined boundaries, at least 40 % of the slot area has to be free. Anyway, with some manual routing and tuning the placement, it will be possible to test some systems. This aspect was expected and therefore one of the requirements for the DRNoC design resources is the occupied area, as it will be explained in further subsection.

3.5.3 DRNoC Design Resources

This subsection describes all the DRNoC design resources that have been created and/or adapted to fit in the DRNoC approach. Since the required resources for building NoCs communication schemes are the most complex, a detailed description of each element can be found after the packet format definition and DRNoC addressing specification. However, first, a few general aspects are included in the next paragraph.

All the resource models are coded in VHDL and some of them, following the hard core design flow that will be proposed in Chapter 4, have been used to get hard cores that have been further grouped in a hard core library, also referred as reconfigurable core library. The main requirement of any model to be included in the system and get a hard core is to have a low area overhead. The core has to occupy less than 60 percent of the assigned slot (slot/core, RNI or RRM), because if this is not possible the efforts needed to have all the core routing inside the restricted slot area are very high and, when the occupancy is around 50 % efforts are high, but it is affordable. This restriction may not be required in future version of the Xilinx design tools if the routing problem is solved.

3.5.3.1 DRNoC Addressing and Packet Format

A packet format is usually specific for a given NoC differs from each other in the header part. Some of them, for instance, include special data that gives routing priority to a packet, while others include all the routing path the packet has to follow. The common

characteristic is that a header always includes the destination address and the size of the packet, just like the DRNoC NoCs packet format. Another common aspect for state of the art NoCs is that the phit size is usually equal to flit size. This is valid for DRNoC NoCs, but with a slight modification further explained.

For understanding the packet format selection, the addressing in the DRNoC system has to be explained and it has to be taken in mind that one of the main goals of the DRNoC approach is to permit the independent communication schemes to coexist and that, the system has to deal with hard cores that can be allocated in any slot/core position. The direct addressing approach in an embedded communication network is to have a unique address for each element and to use this address for data transfer. But on dynamically changing systems, logic and physic addresses have to be differentiated, because hard cores can be reallocated, and this is the addressing approach that has been followed by Pionteck et al., and also the selected one for DRNoC. However, three different addressing layers, described next, (one physical needed for system control and two logical) have been defined for DRNoC and are graphically represented in Figure 3.12:

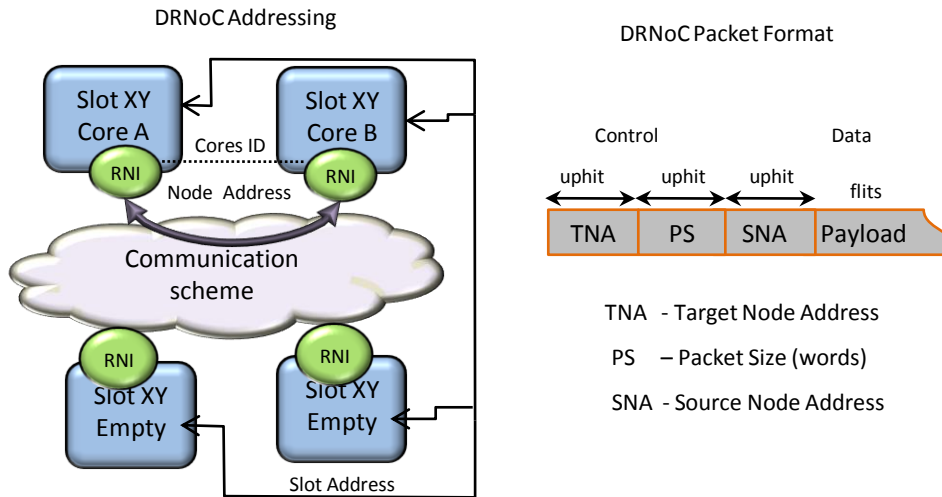


Figure 3.12: DRNoC addressing on the left and the packet format on the right. Three types of addresses are differentiated: Slot address, Node address and Core ID. The packet header part is divided into phit units (uphits), while the payload is divided into flits.

1. **Slot Address** - A unique physical address is assigned to each DRNoC node and to each node element (slot/core, RNI and RRM). A row-column address is assigned to each DRNoC node and two additional bits are used to select which node element is target. For instance in the 4x4 DRNoC architecture, from Figure 3.11, mapped on a 2x16 VA, 6 bits are needed to address all node slots. The 4 most significant bits are related to node row-column coordinates, while the last two are used to select the node element. For example, the address of S:11 is "111100", while NI:11 has the address "111101" and for R:11 it is "111110". Slot addresses are intended

to be used by the control system to manage the slot state.

2. **Node Address** - A unique number that identifies a node in a mapped Communication Task Graph (CTG). In DRNoC, node addresses are directly related to the communication scheme addressing technique, for instance, the XY addressing in mesh NoCs. This permits to create an abstraction for the communication addressing method, that depends on the selected communication scheme and thus, to have a reduced number of addressing bits. For instance, in a 10x10 DRNoC, 7 bits are required for slot addressing, independently from the number of nodes the communication scheme requires. Differently, with node addresses, if the mapped communication scheme in the 10x10 DRNoC is a 2x2 mesh, only two bits are required.
3. **Core ID** - A unique number that is assigned to each core mapped in the system, which is used by cores to communicate between each other. In a dynamic reconfigurable system, there might be more cores in the local core library than currently loaded in the FPGA. Some bits of this ID number are related to the core functionality and others to the mapping, and give the possibility to abstract the core to core communication from the current application it is involved in and from the selected communication scheme.

The DRNoC NoC communication scheme packet format can be seen in Figure 3.12 and, as usual it is composed of a header and a payload part. The header part (that transports the control data) includes the target node address, the amount of data to be sent, measured in words, and the source address. Words have been selected instead of flits because data transactions between two cores are organized in words usually. Like in any other NoC, in DRNoC flits are equal to phit sizes. The flits values used in this thesis are 4, 8 or 16 bits, thus for the worst case where flits = 4 bits, only 16 flits (4 words) could be sent in a packet, while if the measurement is in words, 16 words will be sent.

Another specific characteristic for DRNoC is the introduction of the **Phit Unit - uphit** parameter. This parameter is introduced to give support to the requirement of adaptability in the NoC phit size. Usually, addresses are equal to the phit size. But the phit size in DRNoC can change and applications have to be able to pass from one phit size to another one with the less possible efforts. To solve this problem in DRNoC NoC routers data path has to be independent from the control path (see Figure 3.13) and changes in the phit/flit do not have to affect the header part of the packet and the way routers switching control treat the header. Therefore, the entire packet header is transmitted in uphits and routers switch control is based on uphits. When changing the phit/flit size, the header part of the packet, directly related to the control, remains unchanged. As a result of the uphit introduction, phit sizes are restricted to be: word size \geq phit \geq uphits. If, for instance, uphit=4 and word size is 32 bits, then NoC phit can be 4, 8, 16 or 32. On the other hand, the uphit size restricts the number on nodes that can be involved in a communication scheme. For the previous example, where uphit=4, the maximum number of nodes that can be assigned to a communication scheme is 16 (4 if it is a XY mesh), even if the DRNoC architecture has 100 nodes.

3.5.3.2 DRNoC Router Model

Any router that covers a set of requirements, listed next, could be adapted for DRNoC:

1. The design has to be modular. Buffers, switches and data control have to be strictly identified. This requirement is important for enabling Intra-Core reconfiguration.
2. The entire router has to fit into the RRM size. According to Table 3.1, the real RRM size is 238 CLBs (952 LUTs) for the currently available DRNoC architecture mapping in an XC2V3000 FPGA.

The HERMES network [MCM⁺04b] router covers part of the requirements previously listed and has been kindly provided by the Fernando Moraes and Ney Clazans group from Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS). Indeed, the router is not fully modular, the routing and the arbiter are combined in one module in order to reduce latency. This results in the restriction of no independent routing algorithm and arbitration control reconfiguration. The router has been adapted to fit into DRNoC, and changes made are remarked in order to differentiate the work done, from the original HERMES. Apart from this router, the rest of the designs, included in this section, have been developed within this thesis.

A general view of DRNoC routers is presented in Figure 3.13 and its main characteristics are described next:

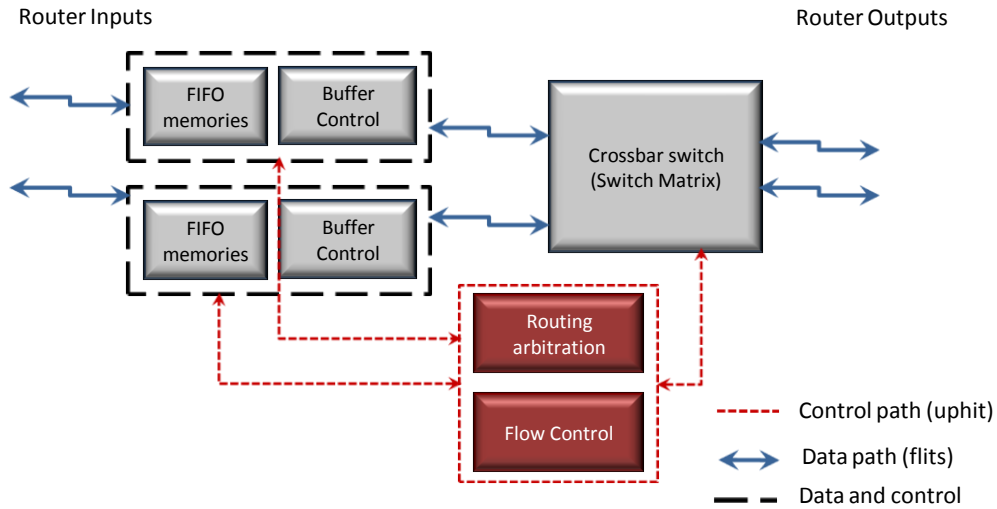


Figure 3.13: DRNoC router general view with highlighted data and control paths.

DRNoC router buffers are allocated at the inputs. Data buffering prior to routing gives flexibility, because buffers do not have to be free if changes have to be made in the routing tables. HERMES FIFO buffers have been fully redesigned to follow separate data and control paths, and have been implemented in the FPGA available BRAMs.

The main idea behind this decision was to be able to dynamically change the FIFO data and address width. Dynamic changes in data width (changing the NoC phit/flit or packet data part), done with Intra-Core reconfiguration, while keeping data buffered in the memory can be done by reconfiguring only the BRAM interconnect columns. The second reason for moving FIFOs to BRAMs was the router size reduction in order to have a better use of the FPGA available resources and to facilitate hard core design (fit onto RRM). Also, because of phit/flit changes, an entire packet has to be kept in the buffer in case a reconfiguration is triggered in the middle of a packet transmission. If this happens the packet that is being transmitted is deprecated from all buffers where it is not complete and retransmitted starting from the point a complete packet is available. Data path changes are possible by using BITPOS that extracts only BRAM interconnect configuration data. However, on board tests have shown that, with the Virtex II memories this technique does not always work properly. In more than half of the performed tests, the expected data to be read from the memory after data or address width changes was not correct. Therefore, although the routers design, the architecture and the tools were prepared to use Intra-Core reconfiguration for data path changes, they have been excluded from the DRNoC reconfigurability in the current version and the used in this thesis router does not include this feature. Anyway, this problem may be overcome with the Virtex 4 and Virtex 5 built in FIFO logic. The counter part of the presented router FIFO design approach is that the actual data buffers size is reduced, and that an additional memory for keeping the packet control data is needed.

The remaining router characteristics are listed next:

1. Routing - Routing tables or XY.
2. Arbiter - Round Robin.
3. Switching - The wormhole HERMES switching has been kept.
4. Interface - The HERMES 4-phase handshake has not been modified.

Regarding routing, two types have been selected (XY and table based) and are described next. It is possible to pass from one type of switch control to the other one by partial dynamic reconfiguration without pulling data out of the buffer.

1. XY routing - This routing is based on the HERMES network routing control, but instead of dealing with phits it deals with uphits.
2. Table based routing - This routing is based on a table that for each destination node provides the number of the output that has to be taken by a packet. Table routing is necessary for irregular NoCs. Tables in the current implementation are mapped to distributed memory. Intra-Core reconfiguration for changing the routing tables can be used and have been tested for small table sizes (up to 4 entries of 4 bits width) by generating partial configuration files with the BITPOS option (the tool will be deeply described in the next section) of extracting only the LUTs content. Tables with higher bit width cannot be mapped in single LUTs, several LUTs are required and this makes more difficult the placement and reconfiguration, since CLBs interconnections have to be also modified. For bigger

tables it is recommended to use Content Accessible Memories (CAM) that could be implemented in BRAMs, but the penalty in the area requirements has to be taken into account. All the CAM control logic is implemented with CLBs in the current CAM cores versions.

3.5.3.3 Traffic Generator Model

Traffic Generators (TG) transmit a certain amount of data to a receiver core. TGs to NIs standardized interface in the current version is based on the AMBA APB protocol. OCP has also been studied as an option, but it requires much more wires than APB, and thus, more communication macros, that results in core real area reduction. It is important to remark that all TGs also receive traffic, answer to request signals, but do not treat the received data. TGs Intra-Core reconfiguration can be used and have been tested in the current implementation to modify TGs target node addresses.

Available TGs are listed next:

1. Regular TGs - This type generates constant, regular traffic to a single or to multiple receivers. Traffic generation is regulated by changing the time interval between two sent packets and the numbers of words that are included in each packet. This approach has been previously used in the HERMES NoC approach in [TMG⁺05], but the provided TGs there are not synthesizable, therefore the main idea have been kept, and behavioral VHDL TGs, suitable for the DRNoC measuring system have been designed.
2. Pareto TG - This type of TGs are common for modeling non uniform traffic. A lot of studies have shown that traffic in multimedia applications is burst based and self similar [VM04]. According to [PGJ⁺05b], such type of traffic can be modeled by mixing ON and OFF periods. The time a traffic generator spends in either the ON or the OFF state is selected according to a distribution which exhibits long range dependence. In the same work, the Pareto distribution has been presented as the most appropriate one for defining the ON and OFF periods. Values, suitable for modeling MPEG 2 video traffic has been also presented in the same work. The Pareto distribution follows the equation $F(X) = 1 - X^{-\alpha}, 0 < \alpha < 1$. For MPEG 2, the ON period is defined to be $T_{on} = (1 - r)^{-\frac{1}{\alpha_{on}}}$ and the OFF period is $T_{off} = (1 - r)^{-\frac{1}{\alpha_{off}}}$, where r is a random number between 0 and 1 and $\alpha_{on} = 1.9$ and $\alpha_{off} = 1.25$. A simplified VHDL implementation taking α values as constant (valid only for MPEG 2) has been designed and included in the DRNoC design resources.

3.5.3.4 Traffic Receivers Models

Traffic Receivers (TRs) implement DRNoC measuring points (MP). TRs MPs include all the needed measuring registers. Two types of traffic receivers have been distinguished depending on the selected measuring scheme: either to perform online measuring (keep the current value), or to keep only max/min values into the internal measuring registers. Each register of an MP is related to a parameter to be measured: latency, the

time a packet header enters the NoC, the time a complete packet has been received in the TR and the number of received data. Each MP tracks data related to a single TG. There may be more than one MP if data for multiple TGs has to be tracked. All this permits to perform a detailed analysis of the system. TRs Intra-Core reconfiguration can be used and have been tested in the current implementation to modify TRs tracked TG Node.

3.5.3.5 Network Interface Models

Several models have been designed for network interfaces (NI) in the TGs side: i) one generates packets for XY network routing, and translates core IDs to XY node addresses and ii) common NIs, that translate core IDs to node addresses. Regarding the TR side, NIs do the inverse address translation and additionally also include measuring points that are the same as the ones included in TRs. For NIs, no Intra-Core reconfiguration has been implemented so far.

3.5.3.6 Other Resources

Some general DRNoC resources have been designed and are needed mainly to create direct links, between two or multiple nodes, and are listed next:

1. For resources to be loaded in RNIs, different feed-troughs, multiplexers/demultiplexers, buffers and dummy logic.
2. For resources to be loaded in RRM, feed-troughs of 2, 3 and 4 inputs/outputs, buffers and dummy logic.

3.5.4 DRNoC NoC Models Generation Tool: DRNoCGEN

A SW tool called DRNoCGEN has been developed as a result from the Master thesis project entitled "Diseño de herramientas de generación de redes en chip para sistemas reconfigurables". The tool, using a set of user defined parameters, automatically generates NoC models from the available DRNoC design resources described in the previous subsections.

This approach is similar to other, like for instance, the MAIA environment presented in [OMP⁺] that generates HERMES NoCs, and the XeNoC system presented in [JFBCR⁺08] that generates complete and synthesizable NoCs from XML descriptions. The main DRNoCGEN difference is that the generated systems, apart from being synthesizable (remember that some HERMES components are not synthesizable) are mapped to the DRNoC virtual architecture, described in the first part of this Chapter. For instance, a user can map a 2x2 NoC mesh, based on XY routing, or a star NoC based on table routing on a defined 8x8 DRNoC.

The tool also permits to create new DRNoC VAs, following the templates presented in Chapter 2, by specifying some parameter values: i) the uphit size, ii) the channel width, iii) the number of subchannels and iv) the number of slots, value directly related to the

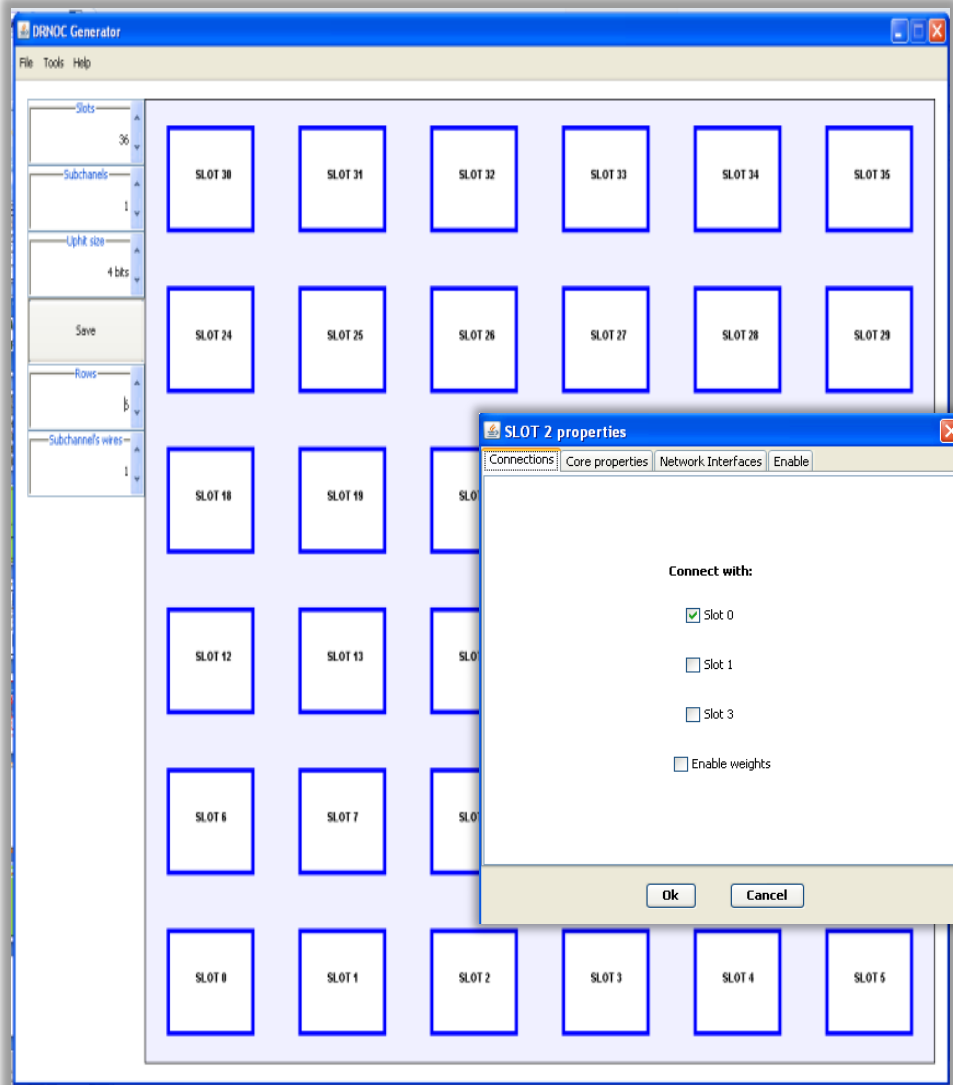


Figure 3.14: DRNoC NoC Models Generation Tool:DRNoCGEN. The DRNoCGEN Tool virtual architecture definition screen shot is shown.

number of nodes in the architecture. The main window of the DRNoCGEN template editor can be seen in Figure 3.14.

Templates, afterwards, are used as a base for building NoCs by specifying the NoC elements: i) NoC nodes (TGs/TRs and NIs), ii) NoC phit size, defined as an integer number of uphits, iii) NoC routers and iv) the buffer size. The tool generates VHDL NoC models, mapped to the DRNoC system following the specified hierarchy and

system templates. The generated top VHDL file includes all DRNoC nodes mapped to a specific slot: cores to slot/cores, NIs mapped to RNIs and all NoC routers mapped to RRM, and also includes all the needed communication macro structures. Additionally the tool can also work directly with system and NoC definition files, instead of defining all the needed parameters by means of the user interface.

3.5.5 DRNoC Reconfigurability

Two types of partial reconfiguration have been distinguished and are supported by DRNoC. As during the previous subsections some restrictions, resulting from performed preliminary tests and analysis, have been pointed out, next a description of each reconfiguration type and a summary of what is currently supported by the system can be found:

1. Intra-Core reconfiguration is defined to change only a certain parameter of a communication element (TG, TR, NIs or routing modules). Currently supported reconfigurations are: i) TR and TG target/source node address, ii) in NoC configuration, change routers routing strategy/arbitration, change router buffers size and change routing tables content. This type of reconfiguration is usually based on small bit manipulation techniques and is related to the network and transaction layers. Due to router restrictions, currently the routing strategy is linked to the arbitration and cannot be reconfigured separately. Other restriction is that a router buffer can occupy only a single BRAM.
2. Inter-Core reconfiguration is used to define the communication strategy by reconfiguring RRM, RNIs and or slot/cores. This reconfiguration type is linked to full slots reconfiguration and affects all the protocol layers implemented in HW. For instance, the used channels width can be modified by changing RRM configuration. As, some problems related to dynamic data path reconfiguration due to FPGAs BRAMs have been reported, data path reconfiguration have been restricted.

3.6 Reconfigurable NoCs Systems Evaluation Parameters

In this section, parameters to be used in different on-chip communication solutions comparison are presented. Parameters have been classified in three groups: i) performance parameters, used to measure the communication strategy performance, ii) structural parameters, used to compare different approaches characteristics and iii) cost parameters, related to the expensiveness of building the required communication strategy. It is important to specify at this point that, the last group of parameters is specific for reconfigurable on-chip communications and has been originally proposed in this thesis.

Each parameter group is described in the next subsections.

3.6.1 Performance Parameters

Performance parameters for communication evaluation are usually based on a standard set of network performance metrics, like bandwidth, latency and throughput [HP03]. In [TPB07], it is stated that the throughput parameter is not appropriate for reconfigurable systems where the communication infrastructure may change. Therefore the parameter of parallelism have been proposed in the same work and defined as the maximum number of independent data transfers that can occur simultaneously. In this thesis, both parameters are considered to be complementary and not exclusive. Throughput is appropriate to characterize the currently implemented communication scheme, and parallelism has been considered as a measuring of the communication diversity but finally it has been excluded as it is not clear how to measure it.

The complete list of parameters with a description of each one can be found next:

1. **Latency (L)** is defined as the time, in clock cycles, between the message first data appearance at the source and the message first data appearance at the destination node. Latency can be measured at the NIs level or at Core level that also includes the network access latency.

The measuring system supports online latency tracking, thus, current latency (CL) is referenced as the latency of the currently received packet.

2. **Throughput (T)** - Some authors define the NoC throughput as the sum of all the link bandwidth [TPB07], others define it as the amount of successfully received data divided by the number of IP cores involved in the transmission and the time needed for it [PGJ⁺05a]. In this thesis, the selected definition is the general one, that is, as the amount of data delivered per time unit to a certain terminal from a network node. It is measured in flits or words per cycle.

The measuring system supports throughput online tracking, thus current throughput (CT) is defined as the amount of data received during a time interval defined between the receptions of two consecutive packets, referred as delta time.

$$T = \frac{\text{total received messages} * \text{message length}}{\text{total transmission time}} \quad (3.1)$$

$$CT = \frac{\text{received messages} * \text{message length}}{\text{delta time}} \quad (3.2)$$

3. **Parallelism** - In [TPB07], parallelism is defined as the maximum number of independent data transfers that can occur simultaneously. Parallelism depends on the number of links k , the number of nodes n , and on the topology. Anyway, this parameter will not be used in this thesis, because a clear method of how to measure or estimate it is not available.

To accurately measure/calculate the performance parameters of throughput and latency, measuring points have to be included following [GIP⁺07] in NIs and TRs.

3.6.2 Structural Parameters

In dynamic reconfigurable systems, structure evaluation parameters, also called structural or physical parameters, are very important. For being fully accurate and provide a fair comparison (the structural comparison can be found in section 3.7.1), the Pionteck et al. definitions of: *Flexibility, Scalability, Extensibility and Modularity* will be used when structural parameters are referred. It is important to remark that the definition used by Pionteck et al. may differ from the meaning of the terms flexibility and scalability normally used in this thesis. Furthermore, other important parameters, namely *Adaptability, Reconfigurability and Quality of Reconfiguration* have been defined and added to the list in this thesis.

The entire list is provided next:

1. *Flexibility* is the ability of a communication structure to support different communication patterns in a fixed design without loss of performance. This parameter is mainly oriented to a fixed design and not to reconfigurability.
2. *Scalability* is the ability of a communication structure to be modified to provide a fixed set of performance parameters independently from the system size and characteristics.
3. *Extensibility* is the ability of an architecture to be extended to larger system designs. Extensibility is closely related to scalability, but it does not require to maintain system performance. This parameter is related to reconfigurability used to extend a communication scheme to include additional cores.
4. *Modularity* the modularity of communication architectures is defined by the extent up to which it can be divided into submodules.
5. *Adaptability* is a measurement of the capability of a communication infrastructure to be adapted in terms of reconfiguration to different communication schemes. In comparison flexibility definition given by Pionteck et al., adaptability strictly refers to architecture changes through reconfiguration and, differently from the scalability, adaptability reflects the system diversity. The lowest level of adaptability is defined as the one that permits to activate or deactivate a point to point link, but the communication scheme itself is maintained the same. The middle level of adaptability adds the possibility of making some changes in the communication architecture, but the main communication scheme is maintained. Like, for instance, to change from a regular mesh to an irregular mesh. Finally, the highest level is the one that supports the creation of a broad set of communication schemes.
6. *Reconfigurability* is a characteristic parameter that measures the capability of a system to support different types of reconfigurations. This parameter is related to the amount of layers that support reconfiguration following the OSI adaptation for NoCs presented in Figure 3.1. The parameter is related not only to the system architecture but also the hard cores architecture design.
7. *Quality of Reconfiguration (QoR)*. The main aspects related to this parameter are: the needed reconfiguration time, the reconfiguration granularity and the system

stability during reconfiguration. The reconfiguration technique provided by the FPGA vendor has a high influence in this parameter, but also the method used for building the systems VAs is important. A low quality of reconfiguration is when the system has to be fully stopped during reconfiguration and then a startup procedure is needed. A middle level is when the system can be reconfigured at runtime/execution time or even real time, but the process is not fully transparent for the system. Finally a high QoR is obtained when the reconfiguration is done at real time and fully transparent.

3.6.3 Cost Parameters

The previous list of parameters is related to the capabilities of a pRTS architecture to be adapted to current requirements and to different communication schemes, but reconfiguration is not for free and additional parameters related to the cost of using exactly that implementation are required. These parameters are important to take a correct decision of which is the best communication strategy in reconfigurable on-chip communication solutions, like DRNoC, and if a reconfiguration is worthy. Parameters are organized in a hierarchical way, taking into account communication structure building elements and also the entire communication scheme. Cost parameter will be used in this thesis during communication schemes evaluation the Chapter 5.

At a single communication building element level:

1. **Channel wire cost** ($C_{cw_{ij}}$) is a relative value that represents to the portion of wires in a single channel used by the j -th channel of the i -th element of the communication scheme over the total number of available wires in a single channel.

$$C_{cw_{ij}} = \frac{N_{wires_{ij}}}{N_{wires}} \quad (3.3)$$

2. **Channel cost** (C_{c_i}) is a value relative to the portion of the amount of the available channels ($N_{channels}$), in the target element position, to be used by the i -th element of the communication scheme.

$$C_{c_i} = \frac{N_{channels_i}}{N_{channels}} \quad (3.4)$$

3. **Area cost** (C_{a_i}) is the faction of the available logic elements N_{le} that are consumed by i -th element of the communication scheme.

$$C_{a_i} = \frac{N_{le_i}}{N_{le}} \quad (3.5)$$

4. **Memory cost** (C_{m_i}) is the fraction of the available memory elements N_{me} that are consumed by the i -th element of the communication scheme.

$$C_{m_i} = \frac{N_{me_i}}{N_{me}} \quad (3.6)$$

All cost parameters at this level are values between 0 and 1. One means that all the available resources will be consumed by the system element. At a higher hierarchical level, taking into account the entire communication scheme, the following parameters have been defined:

1. **Total wires cost** (C_{totalw}) is the sum of all the element wire costs.

$$C_{totalw} = \sum_{i=1}^{N_{elements}} C_{ew_i} \quad (3.7)$$

2. **Total channel cost** (C_{totalc}) is the sum of all the element channel costs.

$$C_{totalc} = \sum_{i=1}^{N_{elements}} C_{c_i} \quad (3.8)$$

3. **Total area cost** (C_{totala}) is the sum of all the element area costs.

$$C_{totala} = \sum_{i=1}^{N_{elements}} C_{a_i} \quad (3.9)$$

4. **Total memory cost** (C_{totalm}) is the sum of all the element memory costs.

$$C_{totalm} = \sum_{i=1}^{N_{elements}} C_{m_i} \quad (3.10)$$

5. **Total implementation cost** (C_{totalI}) is the sum of the total area cost, the total channel cost and the total memory cost.

$$C_{totalI} = C_{totalc} + C_{totala} + C_{totalm} \quad (3.11)$$

Finally, the next parameters define the reconfiguration cost:

1. Number of reconfiguration (N_{reconf}) is the number of reconfigurations that are needed to pass from one communication scheme to another.
2. Reconfiguration cost C_{reconf} could be measured using several parameters. In resource restricted devices, these parameters are related to the portion of the available energy that will be consumed for passing to a new configuration, but if power is not important, then reconfiguration time referred to a full FPGA configuration can be used.

$$C_{reconf} = \frac{E_{reconf}}{E_{available}}, \text{ when power matters} \quad (3.12)$$

$$\text{where } E_{reconf} = P_{reconf} * T_{reconf}, \quad (3.13)$$

$$C_{reconf} = \frac{T_{reconf}}{T_{full}}, \text{ when power does not matter} \quad (3.14)$$

3. Total reconfiguration cost $C_{totalreconf}$ is the sum of all the needed reconfiguration costs.

$$C_{totalreconf} = \sum_{i=0}^{Nreconf} C_{reconf_i} \quad (3.15)$$

The previously listed parameters will be used in the application Chapter, where different communication schemes will be evaluated in order to find the most suitable for being mapped to a DRNoC architecture.

3.6.4 DRNoCs Design Resources Area Requirements

The main idea behind this subsection is to evaluate the area requirements of some DRNoC design resources for being included, following the hard core design flow presented in Chapter 2, in a hard core library. All the results included in this section have been taken from the ISE SW provided by Xilinx after the synthesis process using area optimization.

Synthesis results for different TGs can be found in Table 3.2. The first input on the Table is the Pareto ON/OFF traffic generator, that is the biggest one as it includes, apart from the common logic, all the needed logic for random values generation and Pareto periods calculation. Then, a regular TG that generates uniform traffic for a single receiver and after it, the number of associated TRs has been doubled for the same type of TG, the regular one in this case, but the same is applicable for the Pareto ON/OFF version. From the Table it can be noticed that the influence on the core size of the number of TRs associated to a TG is negligible, and also that, the needed area for all the TG types and the percentage of the used slot covers the 60 percent limit, derived from the restrictions in place and route tools.

Table 3.2: Area overhead and percentage use of the target slot (slot/core) for several Traffic Generators (TGs) and regular traffic generators (TG-MX-Regular) for different amount of target Traffic Receivers (TRs).

TG type	Slice	FF	LUTs	% Slot
TG-Single-Pareto	200	210	290	40
TG-Single-Regular	107	142	177	21.4
TG-M2-Regular	112	143	180	22.4
TG-M4-Regular	117	143	182	23.4
TG-M8-Regular	125	143	190	25

TRs synthesis results are presented in Table 3.3. The Table includes TRs of both available types: i) one follows the max/min value scheme, marked as TR-single-max-min in the Table that includes a single measuring point (tracks data from one TG) and ii) the remaining TRs follow an online measuring scheme and have different amount of measuring points. For instance, TR-M4-online tracks data from four TGs. The same is valid for the synthesis results for NIs that include measuring points associated to TRs

Table 3.3: Area overhead and use percentage of the target slot (slot/core) for two types of Traffic Receivers (TRs) and online TRs (TR-MX-online) that include different number of measuring points.

TR type	Slice	FF	LUTs	% Slot
TR-Single-Max-Min	275	208	490	55
TR-Single-online	286	290	395	57.2
TR-M2-online	489	350	781	97.8
TR-M4-online	810	613	1293	162
TR-M8-online	1571	1120	2522	314.2

Table 3.4: Area overhead and use percentage of the target slot (RNI) for several types Network Interfaces (NIs) for the receiver's side and NIs for and 8 bit flit NoC that include different number of online measuring points (NI-TR-MX-online-8b).

NI-TR type	Slice	FF	LUTs	% Slot
NI-TR-Single-Max-Min-8b	260	208	479	61.9
NI-TR-Single-online-16b	244	326	450	58.0
NI-TR-Single-online-8b	232	328	426	55.2
NI-TR-M2-online-8b	479	350	781	114
NI-TR-M4-online-8b	879	613	1293	209
NI-TR-M8-online-8b	1594	1170	2644	379.5

that can be found in Table 3.4. This table also includes a NI-TR for a 16 bit flit NoC in the first row.

Probably, it would be expected that the area needed for TR-single (no online measuring) is much less than the one for TR-single-online as there is no need of comparators. But special control and double register logic is needed in the online measuring for ensuring correct data reading procedures. Measuring points cannot be accessed while new data is being loaded in the measuring registers (one cycle). Also, once the first register of the measuring point has been accessed by the system, none of the registers can be modified, therefore data is kept in other register that are loaded concurrently in the measuring register at the end of a packet reception. This guarantees the correctness of the measuring point data that is distributed in four registers. From the tables, it can also be noticed that the area needed for a TR or NIs increases linearly with the amount of TGs to be tracked. Even more, none of the cores from TR-M2-online and NI-TR-M2-online-8b until the last one, can be used in the DRNoC system, as none of them cover the 60 percent size requirement (and even, most of them require more area than the available). Therefore in some case, like in the emulation system presented in the application Chapter, it is more suitable to use only simple TR or NI in the design and run the emulation as much times as needed, changing the tracked TG using Intra-Core reconfiguration. However, as it can be noticed from the Table, even these elements are near the 60 % limit.

In Table 3.5, results for two common network interface serializer/deserializer associated to a traffic generator for an 8 bit flit XY NoC and for a 16 bit flit XY NoC can be found. From the Table, it can be noticed that these elements occupation is close to the 60 % limit.

Table 3.5: Area overhead and use percentage of the target slot (RNI) for traffic generators Network Interfaces (NI-TG) for two types of NoCs, 8 bit flit and 16 bit flit.

NI-TG type	Slice	FF	LUTs	% Slot
NI-TG-XY-8bflit	230	320	389	54.7
NI-TG-XY-16bflit	241	337	405	57.3

Finally, NoC routers area requirements will be discussed in the section 3.7.2 (NoC routers Comparison), but here in Table 3.6, DRNoC XY routers for an 8 bit flit NoC with different number of used channels, two and three, are included in order to evaluate their slot utilization. The number of channels does not include the local channel. From the Table, it can be seen that the slot utilization for router is under the 60 percent limit.

Table 3.6: Area overhead and use percentage of the target slot (RRM) for an 8 bit flit XY NoC routers (R-XY) for different number of used communication channels, and a two channels router for a 16 bit flit NoC.

R-XY type	Slice	FF	LUTs	% Slot
R-XY-8b2C	262	183	505	27.5
R-XY-8b3C	300	245	630	31.5
R-XY-16b2C	360	228	690	37.8

3.7 Reconfigurable NoC solutions for Partial Runtime Reconfigurable Systems Comparison

A comparison of reconfigurable communication structure solutions is presented in the next subsections. Solutions from the state of the art, described in the introduction part of the Chapter, are compared with DRNoC. First, a structural comparison of some reconfigurable NoC solutions has been included in subsection 3.7.1. Afterwards, in subsection 3.7.3, a general comparison of all solutions, similar to the 1D systems comparison included in Chapter 2, can be found. In section 3.7.2, DRNoC NoC communication scheme routers performance is compared with other routers.

3.7.1 Structural Comparison

A structural comparison, using the parameters defined in section 3.6.2, of three on-chip communication solutions from the state of the art (DyNoC, CoNoChi and Hübner et al.) and DRNoC can be found in Table 5.17. Part of the data, included in the Table, related to DyNoC and CoNoChi has been taken from [TPB07] (Pionteck et al.), where a comparison of bus solutions, including the Hübner bus discussed in Chapter 2 of this thesis, can be found. The values included in the Table for DyNoC and CoNoChi related to parameters defined by Pionteck et al. have been borrowed and therefore they will not be discussed in the next paragraphs. In this thesis, the Hübner NoC approach has also been included in the comparison, as it is a strategy for providing on-chip connectivity in reconfigurable systems. The rest of the state of the art works (Nollet-Mignollet et al. and Moraes-Calazans et al.) have not been included as they do not directly address on-chip communications reconfigurability.

Table 3.7: Two dimensional reconfigurable systems with Network on-Chip connectivity comparison table. The comparison is based on structural parameters defined in section 3.6.2.

NoC	Flexibility	Scalability	Extensibility	Modularity	Adaptability	Reconfigurability	QoR
DyNoC	low	high	high	high	low	1 layer	middle
CoNoChi	high	high	high	high	medium	1 layer	middle
Hübner	high	high	high	medium	low	1 layer	NS
DRNoC	medium	high	high	high	high	4 layer	middle

Flexibility is directly related to the reduction of the number of the reconfigurations needed due to increasing core features. Including more features in a core could be a great advantage in some cases, but leads to increased area requirement. Regarding DRNoC, its flexibility is indeed the flexibility of the communication scheme mapped on it (NoC, bus, etc). However, in order to achieve a fair comparison, only the NoC communication scheme will be taken into account and thus, the DRNoC flexibility is medium. It is higher than a bus, but is lower compared with CoNoChi because, CoNoChi routers permit packet redirection. Basically, flexibility is defined by the NoC system control and the router architecture, aspects that have not been targeted in DRNoC. The Hübner NoC has high flexibility because it also implements some

additional adaptive routing functionality.

Scalability for NoC based solutions is always higher than for bus based solution and thus it is high for all NoCs. The DRNoC and Hübner NoC **extensibility** is good as both permit new nodes to be added to an already setup communication.

Modularity is high for all solutions but the Hübner et al. one, as all of them are grid based and permit to load cores of any size. Hübner et al. cells can be grouped vertically, but it is not clear if they can be grouped horizontally, therefore its modularity has been considered to be medium. Indeed, the modularity of DRNoC is higher than CoNoChi, as DRNoC differentiates four modules or node elements that can be reconfigured (slot/core, RNI, RRM and SM, three in the implementation because, RRM and SMs have been grouped), while CoNoChi has two (cores and switches).

Adaptability is low for the Hübner et al. approach as it permits only to connect/disconnect a core to the communication structure. It is medium for CoNoChi and DyNoC as both permit to make topology changes, while maintaining the same communication scheme. Finally, it is high for DRNoC, as it permits to completely change the communication scheme.

Reconfigurability for all solutions: DyNoC, CoNoChi and the Hübner is 1 as they permit to reconfigure the physical communication layer (one layer of reconfiguration) and some topology changes can be performed. Differently, DRNoC permits reconfiguration at all the four layers implemented in HW: physical, link, network and transport (see Figure 3.1).

Regarding the **QoR**, the CoNoChi and DyNoC approaches have middle QoR as they use techniques for making possible system runtime reconfiguration and dynamically load cores of different size (the only type of reconfiguration that is supported) and data does not have to be pulled out from the NoC. DRNoC supports broader reconfigurations, but in some cases data does have to be pulled out of the NoC (dynamic data path changes have been excluded from the system). The behavior of the Hübner approach during reconfiguration has not been specified and therefore it is marked with N.S. in the Table.

3.7.2 NoC Routers Comparison

Concerning router area overheads, a comparison of the original HERMES router with buffer depth of 32 flits and flit size of 8 bits, and the DRNoC XY router with the buffers implemented in FPGA BRAMs, also with flit size of 8 bits, are presented in Table 3.8.

All results included in Table 3.8 have been taken from the Integrate System Environment (ISE), provided by Xilinx, after the synthesis process with area optimization. From the table it could be noticed that the DRNoC occupied area, in used slices and Flip-Flops, is around three times less than the original HERMES, but DRNoC routers need a BRAM for each router port.

A comparison of some routers from the state of the art can be found in Table 3.9. The comparison is based on two types of synthesis: for speed and for area optimization. All routers have 8 bit flit size and five ports, including the local one. Notice that DyNoC related results are approximate values. These values have been calculated from DyNoC available area result, accounted in percentage of used FPGA resources in [BA05]. In

Table 3.8: HERMES and DRNoC Routers area requirements

Router type	Num ports	Slice	FF	LUTs	BRAMs
HERMES	2+1	876	882	929	0
HERMES	4+1	1452	1460	1547	0
DRNoC XY	2+1	262	183	505	3
DRNoC XY	4+1	385	296	808	5

the same paper it has not been specified if the presented result are for area or speed optimization, therefore in the Table, the same values appear in both columns. Moreover, for completeness, a DRNoC table router (DRNoC RT) with 5 ports has been included in the same Table. The selected routing tables in this case are 4 positions deep and 4 bits wide (4 uphit NoC).

From the Table, it can be noticed that the DRNoC XY router is the smallest in slices and also the fastest one, compared with the state of the art, but requires the highest number of BRAMs (except DyNoC). It is important to remark that the router version with separated data and control paths needs twice more BRAMs than the conventional one included in the table (still less than DYNoC).

Table 3.9: Some state of the art and DRNoC routers required FPGA area comparison for seep and area optimization.

Router type	Speed		Area		BRAMs
	Slices	Freq[MHz]	Slices	Freq[MHz]	
DyNoC	≈ 409	77	≈ 409	77	≈ 16
CoNoChi	463	90	363	72	4
HERMES	1452	133	1323	83	0
DRNoC XY	391	180	385	129	5
DRNoC RT	444	110	421	85	5

As an example, curves of the influence of number of ports and routing table sizes on the required DRNoC RT area can be seen in Figure 3.15, while the influence of the same parameters in the maximum frequency is shown in Figure 3.16. For both parameters, area increases and frequency decreases with the incrementing of the number of ports. The hops that can be noticed in the curves, when passing from 16 to 32 entries in the tables is due to an increment in the number of nodes (equal to the number of entries) and the needed bits for node addressing.

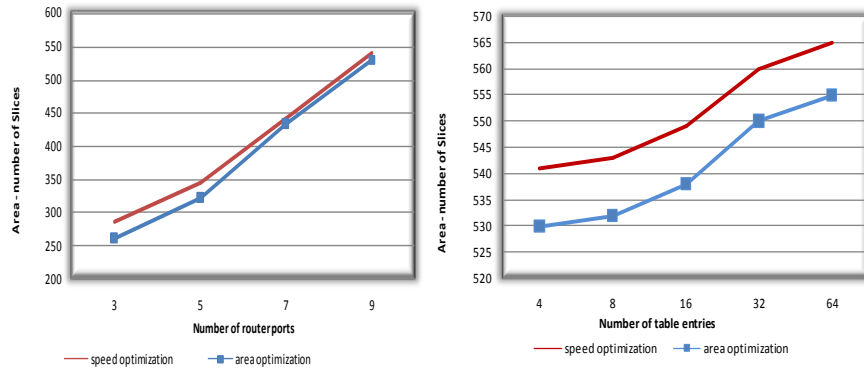


Figure 3.15: DRNoC RT area results for different number of ports on the left and different routing table sizes on the right.

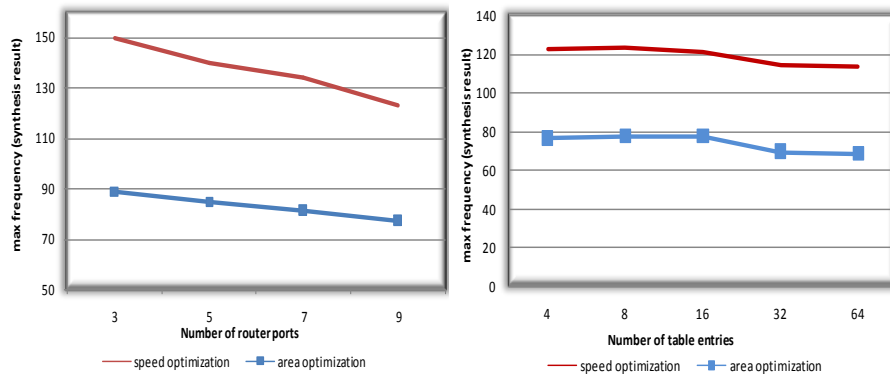


Figure 3.16: DRNoC RT frequency results for different number of ports on the left and different routing table sizes on the right.

3.7.3 NoC based Runtime Reconfigurable Systems General Comparison

A detailed comparison of some solutions, focused on on-chip NoC reconfigurability, has been already presented. Here, a summary of the features of all the solutions can be found in Table 3.10 and a summary of the best implementation results can be found in Table 3.11. The proposed DRNoC architecture has been included in the bottom part of both tables.

Table 3.10: NoC based reconfigurable systems features summary. Summary of all the approaches described in introduction section of the Chapter and the originally proposed DRNoC solution.

Author	Name	Architecture	pRTRS VA model	Cores	Reconfigurability Properties			
					Switches	NIs	Reloc.	Grouping
Nollet-Mignollet et al.	GENKO	Mesh NoC with some reconf. tiles	1D	Yes	No	No	No	No
Hübner-Becker et al.	NS	Cells connected to vertical channels	2D	Yes	SMs	No	Yes	Yes
Bobda et al.	DyNoC	Mesh NoC of reconf. modules	2D	Yes	No	No	Yes	Yes
Moraes-Calazan et al.	ARTEMIS	Mesh NoC with some reconf. cores	1D	Yes	No	No	No	NS
Pionteck et al.	CoNoChi	interconnected 4 port switches	2D	Yes	No	Yes	Yes	Yes
Proposal	DRNoC	Cores connected to a reconf. Xmesh	2D	Yes	Yes	Yes	Yes	Yes

NS-Not Specified.

SMs-Switch Matrices

From Table 3.10 it can be noticed that DRNoC has the widest reconfigurability support compared to the remaining approaches. Concerning the reconfigurability runtime support, the Hübner et al. solution is the only one that gives support in embedded devices, while the Nollet-Mignollet et al. SW support is not related to the HW reconfigurability. From Table 3.11 it can be noticed that DRNoC has the best implementation for Virtex II FPGA, with the highest number of reconfigurable nodes and wider connection links (communication channels). It is interesting to notice that all the presented applications, except from the Nollet-Mignollet et al. multimedia that uses a 1D model, are quite simple. The chosen DRNoC application is an on-chip communication emulation system, where a PC is responsible for the system control. The prepared for testing the emulation system use cases, along with a detailed description of the emulation framework, can be found in the application Chapter. Similarly to the state of the art solutions, the DRNoC emulation framework use cases are quite simple due to current partial reconfiguration technique restrictions.

Table 3.11: NoC based reconfigurable systems. Implementations and applications related features.

Author	Name	Best Implementation Size	FPGA	SW Support	Target Application	Ref.
Nollet Mignollet et al.	GENKO	3x3 with 2 reconf. tiles	VII	NoC TDMA	Multimedia	[MNM ⁺ 04]
Hübner Becker et al.	NS	1x1 slot with 8 cells	VII	Read-modify- writeback	NS	[JBH]
Bobda et al.	DyNoC	3x3 with 1 reconf. module	VII	Jbits BS merging	NS	[BA05]
Moraes- Calazan et al.	ARTEMIS	2x2 with 2 reconf. cores	VII	NS	Reconf. ALU	[MGC ⁺ 07]
Pionteck et al.	CoNoChi	7 switches with 16 bit links	VII	NS	NS	[TPA06]
Proposal	DRNoC	2x2 with 4 reconf. nodes and 40 bit links	VII	Only on Host PC	Rapid Emulation	[YKR06]

NS-Not Specified.

3.8 General Evaluation of the Xmesh Communication Infrastructure

This section compares the Xmesh communication to the 1D bus presented in Chapter 2.

As general evaluation it is important to remark that the technique used for building the Xmesh and the BM-LUTs placement in the FPGA has permitted to allocate a total of 480 wires in 8 FPGA rows and 24 columns (four DRNoC nodes communication channels connection) and, additional 48 wires for the control and measuring system that cross the entire FPGA middle part. The method used to allocate macros for the Xmesh is the same as the one used in the 1D bus based reconfigurable system described in detail in Chapter 2 (nest BM-LUTs), but the amount of wires per row here is higher than the one in the bus based system, because more resources are reserved for the on-chip communication.

Regarding the needed resources compared to the 1D bus based system. A CLB LUT based bus link that crosses the entire FPGA uses less LUTs, a total of 12 LUTs including the access point, than a BM-LUTs Xmesh P2P link that requires 16 LUTs (33 % more).

Finally, the total delay of an Xmesh P2P link is 4.45 ns (2 ns is the delay in the BM-LUTs LUTs and 2.45 is in BM-LUTs connection). On the contrary, the total delay of the 1D bus-v2, reported in Table 2.10 in Chapter 2, is 4 ns for the wires and 1 ns is the delay in the access point LUT, resulting in total delay of 5 ns (12,3 % more compared to an Xmesh link).

As summary, it could be said that the Xmesh connectivity has more flexibility, scalability, is easier to design and has less delay compared to the on-chip communication solution proposed in Chapter 2, but the adverse aspect is the higher area overhead.

3.9 Conclusions

The thesis original contribution, described along the Chapter, is a solution for on-chip communications adaptability, called Dynamic Reconfigurable NoC (DRNoC), that permits to improve the system reconfigurability and flexibility. The main, distinguishing, characteristics of the presented solution are that it permits: i) to define a broad set of ad-hoc communication strategies, heterogeneous, homogeneous, P2P, P2M and/or NoC and ii) several independent communication strategies can coexist in the system. The solution has covered: i) the DRNoC architecture definition that provides the basics for the flexibility and its implementation on Virtex II FPGAs, ii) the supported reconfigurability, along with some implementation restrictions and iii) the design resources, that include proposals of the DRNoC addressing, a NoC router architecture and the NoC packet format.

The core of the solution has been a 2D architecture, designed in Chapter 2 following the method originally proposed in this thesis, that has been adapted to map the DRNoC approach. As an example, DRNoC has been mapped on two FPGAs and the mapping process has been deeply explained. The resulting 2D reconfigurable system has been compared, using a set of parameters, with some state of the art works, described in detail at the beginning of the Chapter. From the comparison, it can be concluded that DRNoC is the most flexible solution and has the widest reconfigurability support. However, improvements are needed in the designed resources (to have better models) and the quality of reconfiguration. Also, not all the defined system features have been included in the final system implementation. For instance, it has not been possible to obtain a secure and stable routers data path reconfiguration. Anyway, the proposed router data path reconfigurability might be possible in future FPGAs. Also, although the used method for VAs definition attempts to reduce the area overhead due to partial reconfiguration, it is still high. Nevertheless a tendency of improving the partial reconfiguration capabilities in the newest FPGAs can be noticed and the achieved results are encouraging.

The overall conclusion of this chapter is that the proposed two dimensional architecture provides connectivity flexibility and scalability to reconfigurable system and the selected application in emulation (to be presented in Chapter 5) is a good application domain for showing some of the advantages of partial reconfiguration.

Hard Core Generation and Manipulation



Design and manipulation of hard cores in flexible and scalable partial runtime reconfigurable systems. Solutions for FPGAs bitstream manipulation.

The use of reconfigurable systems involves the generation and manipulation of the configuration information, or configuration bitstream, associated to the elements that are being reconfigured. For some applications, it is required to be able to do this kind of bitstream manipulation at design time, or even at execution or runtime. The Chapter focuses on the generation and manipulation of FPGAs configuration information and originally contributes in aspects related to tools and design flows for hard core generation and/or partial BitStream (pBS) manipulation.

In section 4.1, a general statement of the goals and features that have to be covered by bitstream manipulation solutions in order to achieve higher flexibility and scalability are presented. After that, in section 4.2 and section 4.3, the related work in this specific topic, divided in commercial and academies solutions, is described in detail. Section 4.4 includes an evaluation of the state of the art along with the proposed in this thesis solution. Then, in section 4.5, a method for bitstream analysis and its application to Virtex II and Virtex II Pro FPGAs, along with a *detailed Virtex II bitstream format description and a set of equations, are proposed*, are presented. Based on the bitstream knowledge and the equations, *two tools and a design flow are originally proposed* in section 4.6. In section 4.7, the proposed solutions are compared with the related the state of the art in section 4.8. Finally, brief conclusions can be found in section 4.9.

4.1 Problem Statement and Goals

An FPGA bitstream file contains an image of the entire device logic elements configuration structured in a certain order. Partial configuration files contain configuration data of specific FPGA logic elements, like, CLB columns, BRAM columns or clock configurations, of already placed and routed designs (hard cores).

In reconfigurable systems, changes and configuration updates are performed by loading new hard cores in the running system. Usually, hard cores for a given reconfigurable system are held in a configuration library and loaded in the partially reconfigurable architecture when needed. This seems to be a simple process, but if flexibility and scalability are targeted, the loading of hard cores in a reconfigurable system is not a trivial and transparent process. When the processes of designing the reconfigurable system and the hard core are considered independent (the main thesis approach), the reconfigurable system itself has to be prepared to consume different type of hard cores (main topic of virtual architectures design in Chapter 2 and Chapter 3) and the cores, on the other hand, have to be prepared for being loaded in different reconfigurable systems (the main topic of this Chapter).

The process of preparing a core to be loaded in a virtual architecture involves hard core manipulation. There are three possible options that have been identified for hard core generation and manipulation (bitstream generation and/or manipulation):

1. Hard cores can be generated as a result of an entire design flow, where the configuration file is created from a circuit net-list.
2. Tools, based on Application Program Interfaces (APIs) that allow manipulating configuration file resources.
3. Tools, based on direct bitstream manipulation that directly access a certain FPGA resource to modify it. The main difference with the APIs based tools is the granularity level at which they work. The granularity in direct bitstream manipulation is usually higher than in the APIs case.

In the next section, some tools and flows from each option will be described and at the end of the Chapter they will be compared with the original solution proposed in this Chapter, which belongs to the third group.

A summary of the main objectives, set for the bitstream manipulation tools in order to follow the thesis main approach are listed next:

1. **Flexibility** - To allow bitstream files (hard cores) to be manipulated. To permit low granularity Intra-Core reconfiguration, coarse grain Intra and Inter-Core reconfiguration and also hard core re-targeting. Re-targeting is the process of changing the target FPGA. This process permits hard cores, designed for one FPGA, to be loaded in a different one. The main aspect related to the flexibility is relocation. The system has to be able to relocate hard cores to as much FPGA positions as possible. Relocation is important because configuration files occupy a considerable amount of memory (file sizes for different hard cores are presented in Figure 4.8) and if relocation is not available, as much images of the hard core, as possible positions in the FPGA have to be kept in memory.

2. Scalability - the solution has to be able to work with different systems and composed of several reconfigurable modules.
3. To be able to deal with as much FPGA resources as possible.
4. Embedded devices are targeted.
5. The solution has to be as fast as possible. The best option would be to have no reconfiguration delay and to perform hard core relocation in several clock cycles, which would lead to real time reconfiguration. However, to have it done in the range of units or tens of ms is a good option that permits runtime reconfiguration, like for instance, to update the system prior to an application execution, or to pipeline an application execution on the same FPGA area. This thesis is mainly focused on runtime reconfiguration for context switching, therefore relocation has to be kept in few to tens of ms.

With respect to design flows, the searched solution has to permit to i) separate the hard core design follow from the detailed characteristics of the target system and ii) to be easy to use, permitting non experts of partial reconfiguration to design hard cores.

It is important to mention that during the thesis when both Virtex II and Virtex II Pro FPGAs are referred, the term Virtex II/Pro will be used.

4.2 Commercial Solutions

In the next subsection solutions related to tools and design flows for partial runtime reconfigurable systems provided by Xilinx are presented. It is worthy to remark that there is also a commercial, non Xilinx, solution provided by the Virtual Computer Corporation (VCC) company. The VCC solutions is a full end-to-end reconfigurable flow that includes a tool for bitstream manipulation for Virtex FPGAs [VCC00], but it has not been updated in the last years and therefore it has been excluded from this state of the art. Nevertheless, it is valuable to mention that the reconfigurable system presented in [FM04] is based on the VCC platform.

4.2.1 Modular Design and BitGen

The Xilinx application note [DL04] describes the exact steps required to successfully design, implement, verify and actively reconfigure a portion of an FPGA device.

The design flow for partial reconfiguration proposed by Xilinx is based on the "Modular Design Flow" described in [Xil05b]. Modular design allows a team of engineers to independently work on different pieces, or modules, of a design, and later, merge these modules into one FPGA design. The main steps of the Modular Design modification, for partially runtime reconfigurable systems, are listed next:

1. *Design entry and synthesis* - To write and synthesize HDL code in conformance with partial reconfiguration guidelines. Xilinx guidelines were deeply discussed in Chapter 2.

2. **Initial budgeting** - Design the floorplan, constrain the logic and create timing constraints for the top-level design and for each module.
3. **Run active implementation** - In this step, mapping, placement and routing (PAR) is done for each reconfigurable module and for each configuration of a particular reconfigurable module. It is strictly important to create a separate project for each module and for each module configuration. For instance, if there are two reconfigurable modules, with two possible configurations, then this step has to be repeated four times and four projects have to be created.
4. **Assembly phase** - This step has two options: i) minimum - full design (initial power-up configuration) and ii) recommended - every possible combination of device configurations of fixed and reconfigurable modules.
5. **Verify design** - Static timing analysis and/or functional simulation for all the possible combinations of reconfigurable and fixed modules.
6. **Visually inspect** the design using the FPGA Editor tool, to ensure no unexpected routing crosses module boundaries. It is very important to be sure that no lines apart from the bus macro ones cross the module boundaries, because otherwise reconfiguration will not be possible. In the latest version of the flow, the discussed later Plan Ahead Flow, this step has been partially excluded when a reconfigurable module is surrounded exclusively with fixed area (however relocation is restricted).
7. **Create full and partial bitstreams** for the design and for individual (partial) bitstreams for each reconfigurable module. The tool BitGen discussed in the following paragraphs is used during this step.
8. **Download device** with initial configuration.
9. **Reprogram** reconfigurable modules as needed with partial bitstreams.

Within the modular design flow, BitGen is used for generating either the full and/or partial BitStreams (pBSs). BitGen takes a routed NCD (Native Circuit Description) file as its input and produces a configuration bitstream.

Another flow, also proposed by Xilinx and used to generate pBSs, is called "Difference-Based BitGen -r Flow". In this flow, users provide two input design NCD files, designating one as the initial configuration and the other as the secondary configuration. BitGen produces a pBS that only assigns values to bits of the FPGA configuration memory that differ in the two files. Unfortunately, it is very unlikely to have two designs equally routed, even if both have similar HDL description and guide files are used. Therefore it is difficult to generate a pBS file using this method that contains exclusively a hard core. This method, according to Xilinx, targets hard core small changes that can be made directly in the NCD file using the FPGA Editor software.

More recently, a new BitGen version, has been released to independently generate pBS files from circuit netlists - the "BitGen PartialMask Flow" [Eto07]. The BitGen PartialMask feature allows users to select which configuration columns of the FPGA

configuration memory will be included in the pBS file and therefore, the granularity permitted by this flow is an entire configuration column. Therefore it cannot be used for large-grain reconfiguration.

Several state of the art design flows for partial runtime reconfigurable systems, based on the described modular design flow have been mentioned in the introduction Chapter.

To summarize, independently of the diverse features, none of the flows proposed by Xilinx, up to now, supports some kind of bitstream manipulation. In other words the bitstream file cannot be modified once generated and therefore it is tightly coupled to the reconfigurable system. The proposed flows do not foresee the possibility of loading a hard core to a different reconfigurable system or even to the same reconfigurable system but in a different position. For instance, if a hard core has to be allocated in a new position in a reconfigurable system, the entire hard core design flow has to be followed. In addition, since solutions for bitstream manipulation in embedded systems, either, are not provided by Xilinx, it is not possible to manipulate and adapt a hard core in order to be loaded it in a reconfigurable system.

4.2.2 JBits

JBits is a set of Java classes, which provide an Application Program Interface (API) to the FPGA bitstream. This interface permits bits, or blocks of bits manipulation on bitstreams either generated by design tools, or read back from actual hardware. JBits can be used mainly in two directions:

1. As a base to develop tools for bitstream manipulation. Some of the most relevant ones will be described in the next subsection, where academic solutions are the main subject.
2. Build circuits by mapping each circuit element directly to an FPGA resource. There are some hard cores developed with JBits, like the ones included in the JBits 2.8 arithmetic RTP cores package, for instance a FIR filter. JBits works at very low level and designers must have very good and detailed, low level, knowledge of the internal FPGA structure. Therefore, JBits is appropriate for small designs, but for large hardware designs it is preferred to work in a higher abstraction level. An example JBits code is shown in Table 4.1.

Table 4.1: JBits pseudo code

JBits pseudo code
<pre> /* Set the Flip Flops Clock */ jBits.set(row,col,S0Clk.S0Clk,S0Clk.GCLK1) /* Disable the CE signal */ jBits.set(row,col,S0CE.S0CE,S0CE.OFF); /* Latch Mode OFF */ jBits.set(row,col,S0Control.LatchMode,S0Control.OFF); /* Set CLB slice0 FF connections */ jBits.set(row,col,S0Control.YDin.YDin,S0Control.YDin.BY); </pre>

Due to the low granularity level, JBits is more suitable for creating software tools that manipulate or extract hard cores, than to describe hardware design. The latest version of JBits (JBits 3.0) supports Virtex II FPGAs. It has been considered as an option to create the tools presented in this Chapter, but it was not selected because, as a Java based tool, it runs slowly, and is too heavy for many embedded devices even if it was possible to port it. JBits require a Java Standard Edition to run. Anyway, there is no JBits version for the state of the art Xilinx FPGAs Virtex 4 and Virtex 5 so far.

4.2.3 XPART

Xilinx, with the JBits experience, has created APIs, implemented in C or C++, prepared to run on an embedded processor and take advantage of FPGA self-reconfiguration. XPART (Xilinx Partial Reconfiguration Toolkit) [BJRK⁺03] currently has methods for CLB related data manipulation and even for relocation of modules composed only of CLBs. However, not only it was never released, but it does not provide a solution for FPGA embedded elements, BRAMs and MULs, therefore modules (hard cores) cannot access such resources. This results in a restricted use of the available resources in the FPGA.

4.2.4 Plan Ahead

Plan Ahead is the latest option for partial reconfiguration, provided by Xilinx in [Xil06a], [Xil06b], [NDG05] and [LBM⁺06]. This tool, based on the Xilinx Modular Design, provides a graphical user interface to facilitate users to deal with different options for the design floorplaning. It is part of the Xilinx "Early Access Partial Reconfiguration Flow (EAPR)" described next:

1. **Design entry** - Define the HDL design description, synthesize it with the conventional Xilinx design software - Integrated System Environment Software (ISE).
2. **Set design constraints** - Define design placement constraints - modules shape and borders.
3. **Implement the non - partial reconfiguration design** - Implement and test the design without partial reconfiguration. This step is recommended for debug and timing and placement analysis.
4. **Implement the base design or top level context** - This is, indeed, the initial budgeting phase of the modular design used to create the top level context.
5. **Implement partial reconfiguration (PR) Modules** - Implement each partially reconfigurable module separately. This is the active module implementation step of the modular design where an independent design is created for each reconfigurable module configuration.
6. **Merge** - The final step in the EAPR design flow is to merge the top, base, and PR modules. During the merge step, a complete design is built from each partial

reconfiguration module and base design combinations. A design has to be created for each combination of a base design and a reconfigurable module configuration. Furthermore, all complete and pBSs are generated using the BitGen tool that has been previously discussed.

7. **Bitstream download and test** - download full and partial bitstream files using the JTAG interface.

The main advance that has been done with the EA PR design flow and the Plan Ahead tool is that, signals (routes) from the base design can cross through a partially reconfigurable region without the use of a bus macro. Although this solution is only applicable to non relocatable modules that are surrounded by fixed area, it shows some progress in terms of partial reconfiguration technique enhancement.

However, Xilinx flows require all the future configurations of the system to be known at design time. Systems, generated with any of the Xilinx flows do not permit the flexibility of supporting updates by loading hard cores that are designed in a different design process or for a similar system. Furthermore, as the EA PR flow is based on the Modular Design, again, bitstream manipulation is not supported and relocation is not possible. Therefore, again, the entire flow has to be repeated if a hard core has to be relocated.

4.3 Academic Solutions

In the following subsections some of the most relevant academic solutions for bitstream manipulations will be presented for the two options: API based and direct bitstream manipulation.

4.3.1 JBits Based Solutions

Some tools, based on JBits are briefly described next. The list included here is not quite extensive, because JBits has been rejected as an option, as it cannot be used in embedded devices:

1. **JBitsDiff** [JRG00] is a Java written tool that directly extracts JBits cores defined in a bounding box (core borders) from a full bitstream. A JBits core is a series of JBits method calls that infer the configuration of the logic within the bounding box. An example JBits pseudo code has been shown previously in Table 4.1.
2. **JBitsCopy** [MDP02] extracts cores from full bitstream file and merge them into another initial full bitstream. Then JBits is used to extract the partial configuration file containing the core. It is worthy to remark that this tool permits core relocation to a different horizontal and vertical position.
3. **JPG Flow and tool** [RS02]. The University of Queensland in combination with Motorola - Australia Software, has created a complete CAD flow based on the JPG tool that generates partial configuration files. The most interesting and distinguishing feature of JPG is that it takes the data needed for hard core

extraction directly from the design constraint file (.ucf file). This results in a better integration with the Xilinx conventional design flow. JPG also includes a GUI (Graphical User Interface) and FPGA loading options.

4. **Core Unifier** is created by the Pontificia Universidade Catolica do Rio Grande do Sul (Brasil) and presented in [MMP⁺]. This tool main feature is to insert and remove a hard core in/from a bitstream file. The tool targets Virtex devices and also has a version based on direct bitstream manipulation, therefore it will be included in the tools features summary Table in section 4.8.2.
5. An interesting approach is the provided by the University of Porto (Portugal) and presented in [SF06b] and [FS05]. They have created a tool, called *BitLinker*, based on a flow that properly combines several pre-build components, selected from a library, to create the hard core to be loaded in the FPGA reconfigurable area. The main, distinguishing characteristic of this approach is that the building components are interconnected during the process of the hard core creation in a similar way to a SW linker. This tool is included in the comparison table in section 4.8 as a representative tool of the JBit based solutions.
6. Another solution is described in [PHP⁺04], where the java-based description language - 'JHDL', from Brigham Young University and its design environment, are integrated with JBits in JHDLBits. The JHDL functionality has been expanded to extract primitives and net information into a data base, suitable to be processed by JBits. Then the bitstream is created by elaborating, placing and routing the primitives in the JHDLBits environment.

4.3.2 Direct Bitstream Manipulation

One of the most remarkable solution based on direct bitstream manipulation is PARBIT (PARTial Bitfile Transformer). The tool has been developed in cooperation between the Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS) and the Washington University.

PARBIT is an open source tool that uses some equations, valid for the Virtex family and made public by Xilinx in [inc03], to identify the position, within a full bitstream, of bits related to an FPGA logic element. The first version of the tool appeared in 2002 and since then, there have been several versions. The last one has been reported in 2005. Its main function is to reallocate and extract Virtex partial bitstreams. PARBIT has three operation modes: slice, block and BIP (Bitstream Intellectual property). In slice mode, the user specifies a slice that contains CLB columns (including IOBs). Data is extracted from a full bitstream file, and a pBS with no relocation is derived [HLK]. In block mode, a rectangular area in the CLB region is specified and extracted from a full bitstream in a target, user defined, position, allowing relocation. In BIP mode, IP cores are re-targeted to a different FPGA device with a reserved area for the BIP core. In PARBIT, logic configuration and routing is specified in an area inside the CLB columns [HL04] and hard cores or modules cannot use FPGA specific resources. In all modes, user options are extracted from an options file. The tool has been used in combination with the DHPs platform described in section 2.3.2.

There are other, more recent, solution for bitstream manipulation that target Virtex II/Pro FPGAS, two of them are listed next:

1. Replica2Pro - This is an interesting solution that permits hard core relocation in 1D architectures (only vertical relocation) [KP06]. The distinguishing characteristic of this solution is that the relocation process is implemented in hardware.
2. CoreUnifier II Pro - Similar lo CoreUnifier, but for Virtex II. The tool can perform vertical and horizontal relocation, but no details or features specifications (if it is based on JBits on a direct bitstream manipulation solution) have been found, and therefore has not been included in the comparison table.

4.4 Analysis of Related Solutions and Proposal

Generally, it can be noticed that the proposed state of the art work do not provide solutions for an embedded reconfigurable system that is already deployed to consume hard cores that are designed in an independent design process. Following the commercial solution, if there is a reconfigurable system that is already delivered and it has to be updated, a specific reconfigurable module has to be designed for it, with precise interface, position and using as a base the design of the deployed system. Otherwise, the update process could result in system malfunction or even damage. On the other hand, there are different bitstream manipulation solutions proposed in the state of the art. The PARBIT solution use, as a base, full configuration files from where a hard core is extracted and loaded in the FPGA or held in a local library. However, dealing with full configuration files in embedded systems results in high memory and time penalties. Furthermore, from the solutions that deal with partial configuration files, the Replica2Pro filter permits relocation only in 1D architectures and the BitLinker, that provides highly flexible features and even some kind of hard core routing, cannot run in embedded systems. As a result, it can be noticed that having a tool that covers all the requirements (set in the beginning of the Chapter) of flexibility, relocation, scalability, on-chip resources usage and that targets embedded systems is not an easy task.

In embedded devices, hard core extraction from complete files is not appropriate due to high storage and execution time requirements. Therefore, in order to gain flexibility and reduce the required memory and execution time, *the approach selected in this thesis, following the idea of having independent system and hard core design processes, is to divide the work in two parts: one deals with the hard core generation or extraction, where a hard core design flow is proposed in section 4.7, and the other one, deals with the hard core modifications/manipulation at execution time on the target system, that will be discussed in the applications Chapter 5.* Each part of the work involves a tool described in sections 4.6.1 and 4.6.2, respectively. As a result, the hard core design and the hard core final allocation are separated. Furthermore, the hard core design flow, originally proposed in this thesis, follows the system templates that are defined in Chapter 2 and reduces the required partial reconfiguration knowledge of hard core designers.

After analyzing the available options for building tools for bitstream manipulation, the best option is to use the technique of direct bitstream manipulation, because it is the fastest one and it is suitable for embedded devices. Both tools, the hard core generation and the hard core allocation one, have to cover the goals of flexibility and scalability, defined in section 4.1.

For applying the direct bitstream manipulation technique, the bitstream format of the selected FPGAs family has to be know. *Since at the time this work started the target FPGA bitstream format was not know at all, a method for studying Xilinx FPGAs bitstreams has been defined and applied for Virtex III/Pro FPGAs. The format has been partially published by Xilinx in [Xil05a] the same year of the first publication of BITPOS (one of the tools presented in this Chapter).*

The bitstream analysis method and the format are described in the next subsections. The parts that are still not public (have not been published by Xilinx) will be remarked.

4.5 Virtex II/Pro FPGAs Bitstream Format

4.5.1 Study Method

The objective of the bitstream study is to determine the bitstream (BS) structure. Two different, complementary, levels have been identified for the bitstream format study:

1. *high-level* - this is the first step in the study. The analysis determines the general format and addressing scheme of the bitstream - which parts of it are assigned to which FPGA component block.
2. *low-level* - once the different BS sections have been identified, it is necessary to analyze the file at a lower level in order to determine which specific bit corresponds to each specific logic cell feature.

4.5.2 High Level Analysis

Any FPGA configuration file is composed of a header part, followed by the FPGA configuration data, and ends with a tail part. The high-level analysis is focused on the structure of the configuration data part of the bitstream (how data is arranged in the BS).

Before starting the analysis, the configuration file word size has to be identified, as well as the beginning of the configuration data. This is marked with a special word that for Xilinx FPGAs, is x'AA995566'.

An approach to perceive FPGA's structure is to see it as a sum of different layers. Each layer is composed of a set of logic elements and/or routing resources. For Virtex II/Pro FPGAs, the logic elements that can be identified are: i) CLBs, ii) CLBs switching matrices, iii) BRAMs/MULs and DCMs, iv) BRAM switching matrices, v) IOBs and vi) clock related resources. An assignment of the listed elements to the FPGA configuration layers, that has been supposed, is as follows: BRAMs/MULs, DCMs and BRAM switching matrices are part of a single layer, CLBs and CLBs switching matrices are part of a second layer and, finally, clock related resources and IOBs belong to a separate layer.

Once the general structure of the configuration file is assumed, and having supposed a layered division, the next steps in the high-level analysis are listed below:

1. First, the smallest configuration unit and what part of the FPGA it configures (this data is public) has to be found. For Xilinx FPGAs, it is a frame. For Virtex II and Virtex II Pro, a frame spans the entire FPGA height, while for Virtex 5 it spans 10 CLB rows.
2. The next step is to find what is the amount of configuration units (frames in Xilinx FPGAs) that are needed to configure a group of elements in a logic layer. For Xilinx FPGAs, a group of elements is a column. This step also confirms or denies the supposed configuration layers division. The process followed for this aim is to assign values to two or more elements in consecutive groups (next

column same row for Virtex II and next block same row for Virtex 5). The generated configuration file data part will be full of '0' except for the assigned values related bits. Then, a simple analysis of the number of words between two visible configurations will give the number of configuration units (frames) in a logic group and will give an idea of the correctness of the layered division. It has been supposed that logic groups, composed of the same amount of configuration units, belong to the same configuration layer. For instance, in Virtex II, CLB and IOBs columns are configured with 22 frames. Therefore, differently from what has been supposed, both elements belong to the same layer. Also, the amount of configuration frames (22) is much higher than the needed to configure the CLB LUTs. A single LUT is configured with 16 bits, therefore as it has been supposed, CLBs and CLBs routing resources belong to the same layer. At this point the order of the logic layers in the configuration file can also be perceived. This step is repeated for all the logic groups, that is, BRAMs, DCMs clocks, etc and as a result it has been confirmed that for Virtex II/Pro FPGAs there are three groups of elements: i) CLB related that include the IOBs and the CLB routing, ii) BRAM content and iii) BRAM interconnect (which is new for Virtex II/Pro, compared to previous Xilinx FPGAs).

3. Once logic groups and logic layers have been identified, it has been supposed that for each logic layer there is a separate, individual, address space. To confirm this, several configuration files that contain identical logic element configuration, but allocated in different groups (columns or blocks) have been generated.

Once the general structure and addressing scheme of the configuration file is known, the low level analysis follows.

4.5.3 Low Level Analysis

The goal of this analysis is to find the meaning of each, single, configuration bit. The required steps are listed next. For the purpose of this thesis, some of them have been automated:

1. Select a logic element as a target.
2. Set or modify one programmable feature of the FPGA (for instance a CLB FF value, or a switch matrix connection). This can be done with the FPGA Editor tool, provided by Xilinx, or directly in the NCD file.
3. Generate the corresponding circuit description file with the FPGA Editor .
4. Generate the corresponding bitstream file in ASCII with BitGen.
5. Analyze the bitstream file to see which bit has been changed.

The results from the Virtex II and Virtex II Pro families BS analysis are described in detail in the next subsection.

4.5.4 Virtex II/Pro Bitstream Format

In this subsection the frame (configuration atomic unit) addressing method, resulting from the high level analysis (now public data) and the meaning of the most important bits of some logic element frames (not public data), resulting from the low level analysis, will be described. This knowledge has resulted from a bitstream study based on the previously described method and using data from [inc03] related to Virtex FPGAs and, the Virtex II format [Xil04] public data at that moment (which was the CRC calculation scheme and the frame size). *Independently from the references used and from the data already published by Xilinx, the bitstream format presented in this section, to our knowledge, is the most complete one.* Data presented in this section will be used afterwards for designing two tools for direct bitstream manipulation.

4.5.4.1 Bitstream Structure

A VirtexII/Pro full bitstream data organization can be seen on Figure 4.1. It is composed in its first part of a header. The header includes, after the synchronization word, a set of configuration data for setting up the device for bitstream downloading. First the configuration memory address to be addressed is written in the Frame Address Register (FAR) and the length of the data to be written, measured in words, in the Frame Data Input Register (FDIR). Then, the configuration memory data is sent in the next order: i) data related to CLBs, ii) data for the BRAM content and iii) BRAM interconnect. The bitstream ends with a tail. After each writing process, a CRC (Cyclic Redundancy Checksum) a check sum is written in a special register (CRC register).

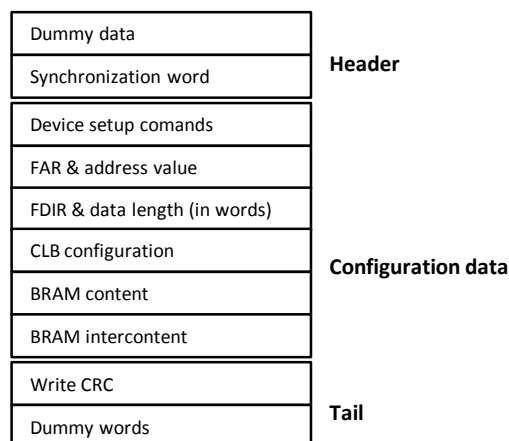


Figure 4.1: Virtex II/Pro full bitstream structure.

4.5.4.2 Frame Addressing

From the high level BS analysis, it is known that there are three logic layers in Virtex II/Pro devices, referred like in Virtex [inc03], as block types: i) BRAM content, ii) BRAM interconnect and iii) CLB related. The RAM content type contains only the BRAM content logic elements while the BRAM interconnect contains the BRAM/MUL interconnect elements and the DCMs configuration data. The CLB type contains all other logic elements (CLB logic, CLB routing resources, Clocks configuration and IOBs).

The Virtex-II configuration logic, like in Virtex FPGAs, is designed so that an external source can have complete control over all configuration functions by accessing internal configuration registers. The Frame Address Register (FAR), for instance, holds the FPGA configuration memory address of the currently written frame. The address is divided in three parts: Block Type (BT), Major Address (MJA) and Minor Address (MNA). The block type field indicates the logic layer. The major address selects a logic element within the block type (logic layer) and the minor address selects a configuration frame of a specific element. In Figure 4.2, the block type, the MJA and the MNA addresses for a Virtex II device can be seen.

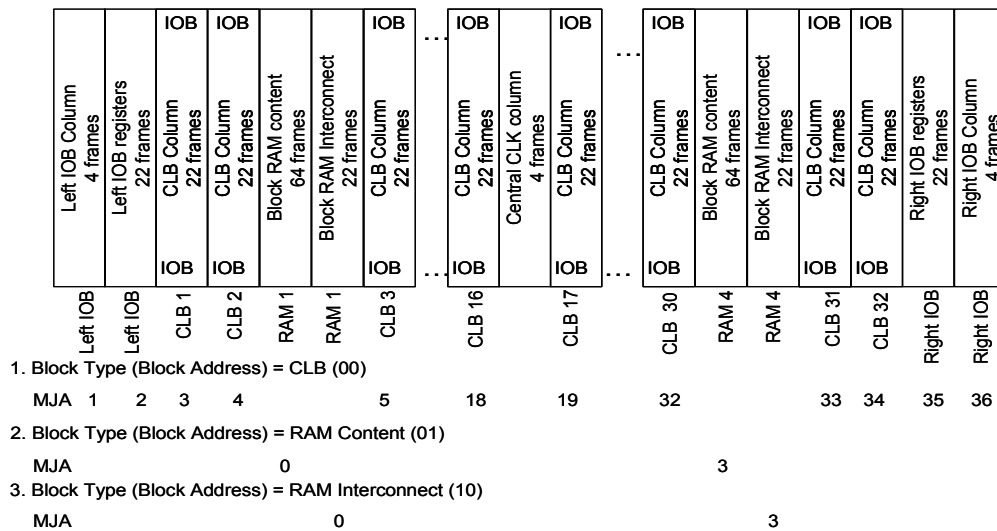


Figure 4.2: Virtex II/Pro bitstream format.

The specific frame format, not published by Xilinx, of each logic element that result from the low level analysis applied to Virtex II/Pro FPGAs can be found in the following points.

4.5.4.3 CLB Frames Format

In Figure 4.3, a schematic view of a CLB configuration is presented. Each Virtex-II CLB includes four slices with two LUTs each. A CLB slice is configured with 160 bits divided in two consecutive CLB frames.

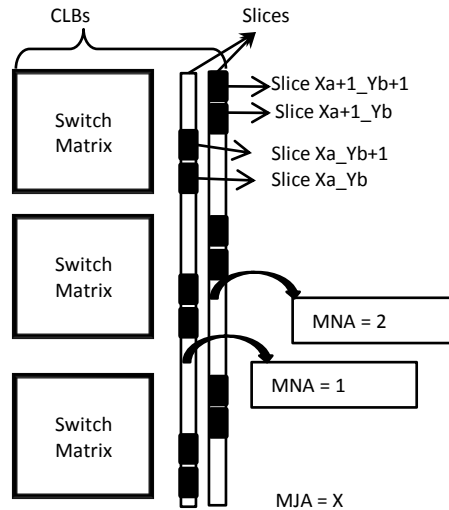


Figure 4.3: CLBs frame addressing. CLB LUTs configuration frames that correspond to frame minor addresses 1 and 2.

The configuration data for the two LUTs (G-LUT and F-LUT) of one slice, differently from Virtex devices, is allocated in a single frame. The configuration data for slices $XaYb$ and $XaYb+1$, see Figure 4.3, appears in the CLB configuration frame with $MNA = 1$ in consecutive positions, while configuration bits for slices $Xa+1Yb$ and $Xa+1Yb+1$ appear in the frame with $MNA=2$.

The format of an entire configuration frame for the CLB elements group can be seen in Table 4.2. The first 16 configuration bits, usually used as pad bits in Virtex FPGAs, here are related to the clock vertical distributions along CLBs.

Table 4.2: CLB frame format

Element	bits
Pad & Clock	16 bits
Top IOBs	80 bits
CLB.Colx.RowN	80 bits
CLB.Colx.RowN-1	slice 40 bits
	slice 40 bits
..	..
CLB.Colx.Row1	80 bits
Bottom IOBs	80 bits
Pad & Clock	16 bits

Regarding slice configuration bits, each slice is configured with 40 bits. The meaning of almost all bits is presented in Table 4.3. It is worthy to highlight that the sixteen bits needed to configure one LUT are stored inverted, while Flip-Flop data is stored in their true sense and when reading or writing LUT data from/to the bitstream, the stored

Table 4.3: CLB Configuration Bits

Element	Bits	Note
F LUTs	15-0	F LUT configuration - LSB bit first
INIT0/FFX	17	Selects GND or VCC as INIT value for FFX
RAMX	18	RAM Enable
INIT0/FFY	19	Selects GND or VCC as INIT value for FFY
MODERAM	21-20	RAM mode (10-shift, 01 RAM)
RAMY	22	RAM Enable
G LUTs	24-39	G LUT configuration - MSB bit first

value is the negated. For example, a 4-input AND gate has the truth table LUT[15:0] = 1000000000000000. This truth table is stored internally in the LUT as LUT[15:0] = 0111111111111111. Another aspect is that data for the G-LUT is stored with the Most Significant Bit (MSB) first while for the F-LUT it is with the Least Significant Bit (LSB) first.

4.5.4.4 IOBs Frame Format

The IOBs and the CLB group of columns are configured with the same amount of frames and both elements belong to the same configuration layer (block type). It seems, taking into account the similar structure of the IOBs and the CLB block, as well as other analysis, that the IOBs and the CLBs are configured with the same amount of bits - 80 per frame. So the frame organization for the IOBs is the same as the CLB frame and will not be further discussed.

4.5.4.5 BRAM Content Frame Format

The BRAM content frame format is shown in Table 4.4. The 16 Kb needed to configure one BRAM are distributed in 64 frames. There are 320 bits related to a BRAM in each configuration frame, that is 4 times the needed for CLBs (one BRAM column occupies the area of 4 CLB rows). The low level analysis has also covered BRAM content bits ordering in the BS, resulting in a special ordering scheme applied to the initial vector defined in the HDL code or the FPGA Editor. The ordering scheme is too extensive and not relevant for the tools presented in this Chapter and therefore has not been included.

Table 4.4: BRAM content frame format

Element	bits
Pad & Clock	16 bits
Top DCM	80 bits
BRAM.Colx.RowN	320 bits
BRAM.Colx.RowN-1	320 bits
..	..
BRAM.Colx.Row1	320 bits
Bottom DCM	80 bits
Pad & Clock	16 bits

4.5.4.6 Switchboxes Analysis

The low level analysis has also been applied to the FPGA switch boxes, resulting in the knowledge of a considerable part of the configuration bits, but resulting data are too extensive to be included in this document.

Regarding Virtex II Pro FPGAs, the presented CLB and BRAM frame format is maintained for the area not used by the embedded Power PCs (PPCs). PPCs occupy 16 CLB rows, several CLB columns and two BRAM columns. Some configuration bits related to this area are used to configure static inputs and switch matrixes associated to the PPCs.

4.5.5 Bitstream Access Equations

As a result of the knowledge of the BS format, a set of equations for accessing a certain element in a full Virtex II/Pro configuration bitstream have been originally defined and are presented next. For better understanding, some definitions to be used are included in Table 4.5. For compatibility with older FPGAs, the definitions have been kept as similar as possible to those used for Virtex based FPGAs in [inc03].

Table 4.5: Definitions

Term	Definition
<i>MJAX</i>	Element mayor address ($X=BRAM$ o CLB)
<i>MNA</i>	Element minor address
<i>bpf</i>	Device bits per Frame
<i>Chipcols</i>	Total number of device CLB columns
<i>Chiprows</i>	Total number of device CLB rows
<i>RAMcols</i>	Number of device RAM columns
<i>RAMrows</i>	Number of device RAM rows
<i>Xrow</i>	Element ($X = CLB/BRAM$) row index
<i>Xcol</i>	Element ($X = CLB/BRAM$) column index
<i>fridx</i>	Searched frame start index within a full bitstream
<i>bitidxst</i>	Element start bit index within a frame

CLB frames identification:

$$fridx = 8 + [(MJAc_{lb} - 1) * 22] + MNA \quad (4.1)$$

$$MJAc_{lb} = CLB_{col} + 2 \quad (4.2)$$

BRAM content frames identification:

$$fridx = 12 + [(chipcols + 2) * 22] + [(MJAb_{ram} - 1) * 64] + MNA \quad (4.3)$$

$$MJAb_{ram} = RAM_{col} - 1 \quad (4.4)$$

BRAM interconnect frames identification:

$$fridx = 12 + [(chipcols + 2) * 22] + [(RAMcols) * 64] + [(MJ Abram - 1) * 22] + MNA \quad (4.5)$$

Values given by these equations represent offsets in words in a full, uncompressed bitstream, starting at the data part beginning immediately after the FDIR writing command.

To provide more flexibility and smaller granularity, the next two equations identify a logic element within a frame. They can be applied to a full configuration file once the target frame has been found or to partial configuration files.

Identification of a CLB element within a CLB frame:

$$bitidxst = fridx + [(CLBrow - 1) * 80] + 96 \quad (4.6)$$

Identification of a BRAM element within a BRAM frame:

$$bitidxst = fridx + [(RAMrow - 1) * 320] + 96 \quad (4.7)$$

Other equations could be deduced from the presented bitstream format, like searching for a specific slice, LUT or BRAM configuration bit. Here, only the ones used by the tools described in the next sections have been included.

4.6 Direct Bitstream Manipulation Solution for Virtex II/Pro FPGAs

The equations described in the previous section have been integrated in *two tools for direct Virtex II/Pro bitstream manipulation, originally proposed in the following two subsections*.

The main common features of both tools are listed next:

1. Both deal with CLBs and BRAM/MULS elements.
2. Are valid for 1D and 2D virtual architectures.
3. Permit vertical and horizontal relocation. The technique used for hard core relocation is to change MJA addresses, like in all tools based on bitstream manipulation.
4. Valid for all Virtex II and Virtex II Pro FPGAs.

4.6.1 BITstream POSitioner (BITPOS)

BITPOS stands for *BITstream POSitioner*. The tool is intended to be used during hard core design. Its main function is the generation of a partial bitstream file that

corresponds to a hard core. This process is done by extracting the hard core from its basic framework, a full configuration file, which will be called a *hard core repository*.

The main feature of the tool is that the target position of the hard core to be extracted can be defined by the user (relocation). The BITPOS relocation property is important at design time for testing dynamically reconfigurable systems. It is important to extensively test reconfigurable systems by performing several partial reconfigurations with different cores, and allocated at different position in the FPGA. With the PBITPOS tool this process is straightforward, while with the Xilinx solution, the entire design flow has to be repeated.

Another property of BITPOS is that a user can avoid the definition of a target hard core position, resulting in hard cores that are not linked to a specific position in the FPGA (partial configuration file with FAR value equal to x"00000000"). This feature is valuable when the hard core target position is not known at design time and the system security requirements are not high.

For achieving a correct hard core extraction, the position of the hard core within the core repository has to be known. The position is specified by row/column coordinates written in a *hard core description file*. An example of the values included in a description file are presented in Table 4.6 and a graphical representation of hard cores coordinates can be seen in Figure 4.4. This approach is similar to the used by the PARBIT tool as both are based on direct bitstream manipulation.

Table 4.6: Description (desc) file values. These files are used to define the core initial and target position. Some of the values included in the example are not obligatory, like the initial position.

Parameter	value
XFPGA	XC3V3000
startlbccl	3
startlbcrow	5
endlbccl	7
endlbcrow	81
startlramcol	1
startlramrow	2
endlramcol	4
endlramrow	15
iMPACT	1
writefullheader	1

For not losing generality and for being totally accurate with the slots definition from the previous Chapter, CLBs and BRAMs are treated separately; therefore each element (CLBs and BRAMs) has its own coordinates in the description file, see Table 4.6.

BITPOS has two operating modes:

1. In *Simple mode*, a hard core is extracted from its repository and allocated in the target position specified in the description file. If there is no need of relocation, the target and current values will be equal. This mode is used when a core occupies the entire FPGA height - virtual architectures based on 1D slot distribution models.

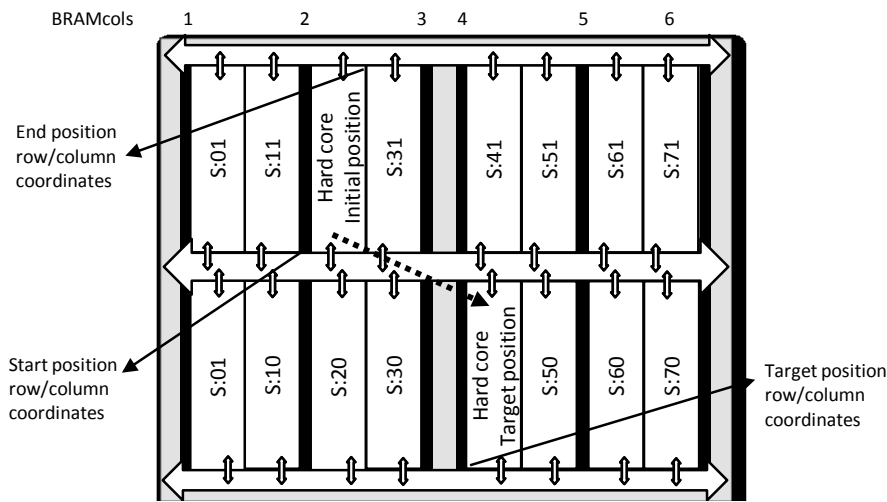


Figure 4.4: Hard core example initial and target position. The example is based on a virtual architecture with 2D slot distribution presented in Chapter 2.

2. **Merge mode.** This mode is used when a core is restricted in a bounding box. All the outside the upper and lower core borders is extracted from another bitstream file. This mode permits hard core relocation in 2D reconfigurable systems or in 1D where the communication channel is defined outside the core boundaries, like the bus based ones defined in Chapter 2.

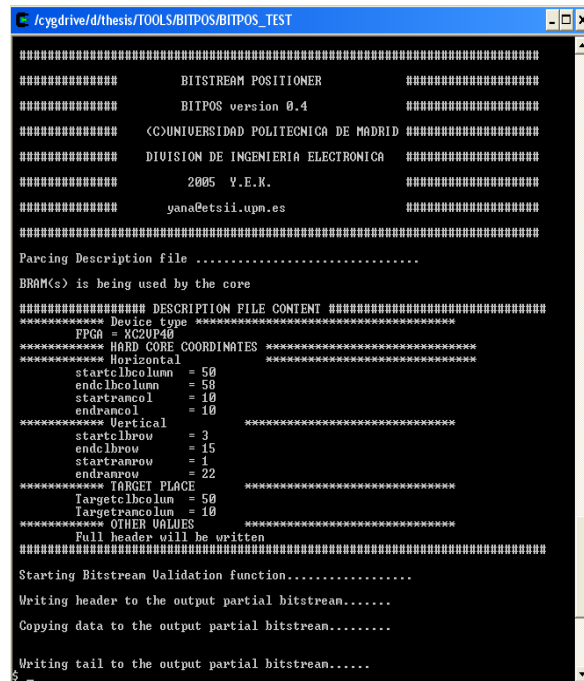
An example BITPOS execution window can be seen on Figure 4.5, and other tool features are listed next:

1. It can extract only the BRAM content data (-k option). This feature is oriented to change only data saved in the on-chip memory. For instance, to modify some hard core coefficients held in a local memory while keeping the hard core functionality.
2. Extract only CLB LUTs configuration frames (-l option). This option is basic for small bit manipulations. For instance, to modify small hard core parameters, which are held in LUTs instead of BRAMs.
3. Extract a hard core without specifying its position (-p option). Such files cannot be directly downloaded into the FPGA. This option is valuable when the target hard core position is not known at design time and security requirements are not high

The BITPOS algorithm works as follows: It first performs a bitstream validation procedure to guarantee that input bitstream(s) fulfill the standard. Then, data from the description file is extracted and some basic parameters are compared with the data read from input bitstream(s) to check values correctness. After that, indexes of the search data within input file(s) and new configuration address values are calculated using the equations presented in the previous section. In the next step, the header of the output pBS is written. Next all the hard core related data is copied column by column. Each

column is addressed separately, using the target hard core position, if it is specified, and the corresponding CRC values are calculated with the scheme presented in [Xil04] and written in the file. Finally, the bitstream tail is written.

The same calculation technique is applied for all the configuration layers (CLBs related, BRAMs/MULs content and BRAM/MULs interconnect) separately.



```

##### BITSTREAM POSITIONER #####
##### BITPOS version 0.4 #####
##### <C>UNIVERSIDAD POLITECNICA DE MADRID #####
##### DIVISION DE INGENIERIA ELECTRONICA #####
##### 2005 Y.E.K. #####
##### yana@etsii.upm.es #####

Parsing Description file .....
BRAM(s) is being used by the core

##### DESCRIPTION FILE CONTENT #####
##### Device type #####
FPGA = XC2UP40
##### HARD CORE COORDINATES #####
##### Horizontal #####
startlbcolum = 50
endlbcolum = 58
startrancol = 10
endrancol = 10
##### Vertical #####
startlbcrow = 3
endlbcrow = 15
startrancrow = 1
endrancrow = 22
##### TARGET PLACE #####
Targetlbcolum = 50
Targetrancolum = 10
##### OTHER VALUES #####
Full header will be written

Starting Bitstream Validation function.....
Writing header to the output partial bitstream.....
Copying data to the output partial bitstream.....
Writing tail to the output partial bitstream.....

```

Figure 4.5: BITPOS execution window print screen image. Execution in merge mode with verbose output activated. The parsed description files options values can be seen in the figure.

4.6.2 partial BITstream POSitioner (pBITPOS)

pBITPOS, stands for *partial BITstream POSitioner*. The tool has a similar algorithm to BITPOS, but pBITPOS is prepared to deal with hard core relocation. The main feature of the tool is that it is meant to work in embedded systems. An example application is presented in the application Chapter 5, where pBITPOS will be integrated in a remote reconfigurable device.

pBITPOS uses as target file any partial bitstream, generate by BITPOS or read-back from the FPGA, as long as it fulfills the partial configuration bitstream format defined by Xilinx and is an uncompressed file. The output of the program is a new merged and/or relocated hard core containing the core related configuration data, the calculated new addresses and the configuration header and tail.

A specific option of pBITPOS is the re-target mode that permits to load a hard core on a different FPGA. pBITPOS, for instance, permits an already delivered reconfigurable

system not only to consume hard cores that are not specifically design for that system, but also hard core that initially targeted a different FPGA. A pBITPOS execution screen in this mode can be seen in Figure 4.6. It is important to clarify, that this feature has been firstly included in the PARBIT tool, where the possibility of changing from a smaller to a bigger Virtex FPGA has been provided. pBITPOS permits to change the hard core target FPGA from a Virtex II family devices to a Virtex II Pro family FPGA.

```

Cygdrive/d:/thesis/TOOLS/pBITPOS/pBITPOS_TIME/Virtex_II_pro
$ ./pBITPOS_U05.exe -v xc2vp30_3.desc xc2vp30_p3.bit out.bit xc2vp30_p3.bit
Executing pBITPOS in complex mode, complementary file = xc2vp30_p3.bit.....
=====
ENAMORADO partial BITSTREAM POSITIONER
pBITPOS version 0.4
UNIVERSIDAD POLITECNICA DE MADRID
DIVISION DE INGENIERIA ELECTRONICA
V.E.K. 2005
=====
sync word found!!
Parsing Description file .....
BRAM(s) is being used by the module
=====
DESCRIPTION FILE CONTENT
=====
Device type
FPGA = XC2VP30
MODULE COORDINATES
=====
Horizontal
startclbcolumn = 1
endclbcolumn = 4
starttrancol = 1
endtrancol = 1
=====
Vertical
startclbrow = 3
endclbrow = 16
starttrancol = 2
endtrancol = 2
=====
TARGET PLACE
targetclbcolumn = 1
targettrancol = 1
targetclbrow = 0
targettrancol = 0
=====
OTHER VALUES
heightclb = 13
heightbram = 0
Full header will be written
Starting Bitstream Validation function.....
Writing header to the output partial bitstream.....
Copying data to the output partial bitstream.....
Writing tail to the output partial bitstream.....
$

```

Figure 4.6: pBITPOS re-target mode execution window print screen image. The parsed description file option values can be seen in the figure.

Re-targeting is possible when the number of target FPGA rows and columns are equal, or greater than, the hard core ones. For this aim, the tool has to know the target FPGA rows, and therefore an additional parameter is added in the description file (TFPGA - Target FPGA).

Another pBITPOS advantage, is that it can work with up to three input pBS files and combine them to generate the new FPGA configuration (similar feature to BitLinker, but for embedded devices and more restricted). This option provides wider flexibility when dealing with 2D slot distribution architectures when there are two or more hard cores allocated in one column.

4.7 Design Flows

Two design flows will be described in the next two subsections: i) one used for hard core design and ii) the second is used for hard cores adaptation. These flows have been used during hard cores design for most of the reconfigurable systems that will be presented in the application Chapter 5.

4.7.1 Hard Core Design Flow

The main difference of this *originally proposed hard core design flow* with respect to the state of the art is that, i) it is based on architecture templates, ii) it separates the system architecture design process from the hard core generation and that, iii) it includes relocation capabilities.

The hard core design flow follows a template based approach, starts with a system architecture template selection, and permits several hard cores to be design in parallel (following only once the flow). Thus, the hard core design process is simplified and it does not have to be repeated several times to get the needed configuration files (as is the case of the Xilinx Early Access Partial Reconfiguration Flow - EA PR, where some steps have to be repeated once per reconfigurable module). Also, due to the template based approach, hard core designers do not need to have extensive reconfigurability knowledge (although it is still obligatory for virtual architecture designers). Anyway, hard core designers still have to check slot boundaries after the place and route process (restriction that results from the deficiencies of the Xilinx place and route tools). Moreover, as the proposed flow uses BITPOS, it does not have to be repeated in order to allocate a core in a different position during the system design process. Differently any Xilinx flow has to be repeated to get the core reallocated.

The design flow steps are summarized in Figure 4.7 and are described afterwards:

1. The first step of the hard core design is to estimate: i) the hard core communication protocol requirements - number of wires and on-chip interconnections and ii) hard core size. With this knowledge, a suitable virtual architecture is selected. It is recommended to select an architecture with more communication wires and larger slot sizes than the estimated a priori. As a result, architecture definition files are selected and the hard core design can begin in the next step.
2. Hard cores are designed using the architecture hard core template in any design environment (hard core templates are a part of the virtual architecture definition files described in Chapter 2). This step includes all the required core functional validation. The default position of the hard core in the architecture is defined by instantiating (mapping) the hard core in the appropriate slot in the higher level system template file. If no other slots from the architecture will be used, then all slot instances will remain with the default empty cores mapped to them. Furthermore, several cores can be designed in parallel by selecting an architecture with a suitable number of slots.
3. Next, a full configuration file containing the system architecture definition and the hard core(s) is generated. For this step, any design flow can be used: the FPGA

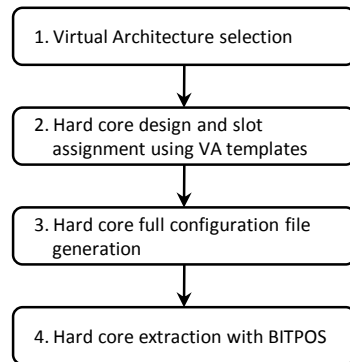


Figure 4.7: Hard core design flow. The flow starts with the selection of an architecture template, which results from the virtual architecture design method steps. The flow also uses the BITPOS tool presented in this Chapter.

conventional design flow (ISE based), the Plan Ahead design flow or the Modular Design. Some problems in using the Plan Ahead design flow have been noticed. It has not been possible to integrate custom communication macros, like the macros presented in Chapter 2, where long wires cross the entire FPGA. Differently, the flow has been successfully used with systems based on short (neighboring) communication macros. On the other hand, the classical command line Modular Design flow has been successfully tested for all cases. Anyway, to keep the design flow simple and use only one design tool, the ISE SW has been used for hard core design. A distinguishing characteristic of this flow from the Xilinx ones is that with the proposed flow, a single project is created for each hard core design and even more, several hard cores can be created in parallel with a single project.

4. The last step of the flow is to extract or generate hard core(s) from the full configuration file. BITPOS is used in this step and, depending on the target use of the hard core, it can be generated with a specific position: with the same initial position or without specifying the hard core position. The use of the latest versions of the BitGen or EA PR flows may also permit to directly generate hard cores from ncd files, instead of using BITPOS. However, in systems where a core can be allocated in different positions, it is recommended to test hard core relocation before including the core in the library or repository. This relocation with BITPOS is straightforward. Differently, with Xilinx tools, the entire design flow (EA PR or Modular Design) has to be repeated for each position.

Once hard cores are designed and tested, they can be held in a local or a remote library for further integration in a reconfigurable system. The generated hard cores can be integrated in any virtual architecture as long as there is enough room, the target FPGA is compatible (bitstream adaptations possible) and the core interconnection is supported

by the defined on-chip communication infrastructure.

4.7.2 Small Granularity Reconfiguration

When small granularity reconfiguration is foreseen in the target application, a virtual architecture is not needed as it was already pointed out in Chapter 2, during the description of the first step of the virtual architecture design method. In this case, the flow described next, based on the Xilinx small bit manipulation solution and BITPOS, is recommended:

1. First, the hard core structure has to be organized or reorganized in a modular way. The sections to be reconfigured have to be strictly separated, in individual components, and with clear, defined, interfaces. This restriction is not applicable when the part to be partially reconfigured resides in BRAM memory. Also, the position of the reconfigurable sections has to be strictly known and fixed. This step can be executed using the floorplanning tool provide in Xilinx ISE.
2. After that, the standard design flow is followed. It ends with a full configuration file generated by BitGen.
3. Then, BITPOS is used to extract a section of the hard core. If the section is allocated in LUTs, then BITPOS extracts only the two configuration frames related to LUTs, while if it is in BRAMs, then BITPOS extracts only the BRAM content configuration frames.

Furthermore, if a new value is going to be assigned to the same core section (registers, LUTs or memories), the FPGA Editor can be used to directly edit the routed design and then BitGen can be used to directly generate a full configuration file without re-placing and re-routing the design (Xilinx small bit manipulation flow). Afterwards, the previous step is repeated to have the hard core new section pBS.

4.8 Results

This section presents the results of the tools evaluation, that is related to execution time analysis and a comparison of the general features of some tools and flows from the state of the art and the ones proposed in this Chapter.

4.8.1 Hard Core Benchmark Library

Regarding the tools evaluation, apart from the tools features that are summarized in Table 4.7 at the end of the Chapter, an important aspect, set as requirement in the introduction section of this Chapter, is execution time.

For measuring the tools execution time, a benchmark has been defined. The benchmark includes 95 hard cores of different sizes for Virtex II and 60 hard cores for Virtex II Pro FPGAs. All the 155 hard cores are composed of different numbers of CLB columns

and access to different number of BRAMs, up to four BRAM columns (the core size is marked as CLB/BRAM column in the X axis of all following figures). Additionally, around 20 hard cores that do not use BRAMs have been included in the benchmark to see the influence on the use of BRAMs in the execution time. Some hard core files sizes from the benchmark XC2V3000 FPGA are included in Figure 4.8. The included hard cores are composed of different number of CLB and BRAM columns (for instance 12/02 is a hard core composed of twelve CLB columns and two BRAM columns) and also the full configuration file can be found. Hard core sizes go from 256 to 7168 slices and the file size goes from 91 KB to 591 KB. Differently, a full configuration file requires 1282 KB.

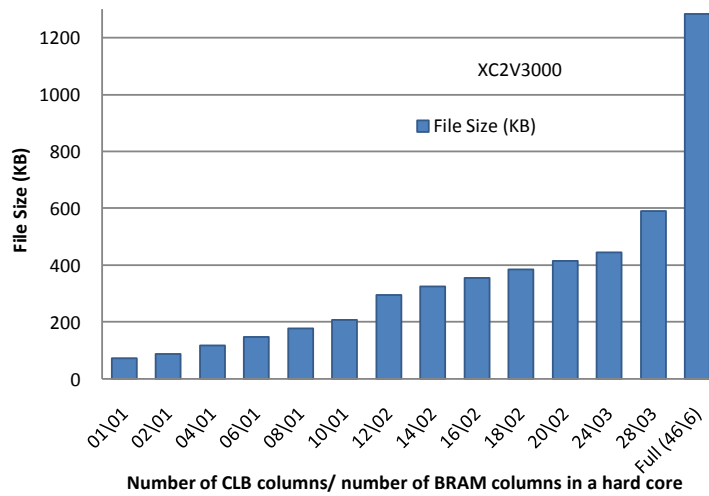


Figure 4.8: Different hard cores file sizes, from the smallest that contains a single CLB column and a single BRAM column (01/01) to a full bitstream.

4.8.2 BITPOS Execution Time Results

BITPOS execution time is measured as an average value of 100 consecutive executions on an Intel Pentium Centrino - 1.7 GHz. The BITPOS execution time in *merge mode* for generating the benchmark hard core from the respective hard core repository is presented in figure 4.9 for Virtex II devices and in Figure 4.10 for Virtex II Pro. Merge mode has been selected because it is supposed to be more time consuming.

As it could be expected, the execution time grows proportionally with hard core size for all FPGAs. Execution time is very high for larger hard cores, reaching values of 7 seconds for Virtex II Pro hard cores that occupy around 50 % of the FPGA CLBs and 30 % of the BRAM columns (see Figure 4.10). Common, small and medium size hard cores, like those that will be mostly used in this thesis, occupy typically from 4 to 12 CLB columns and one or two BRAM. For these cores, execution time is kept under 3 seconds.

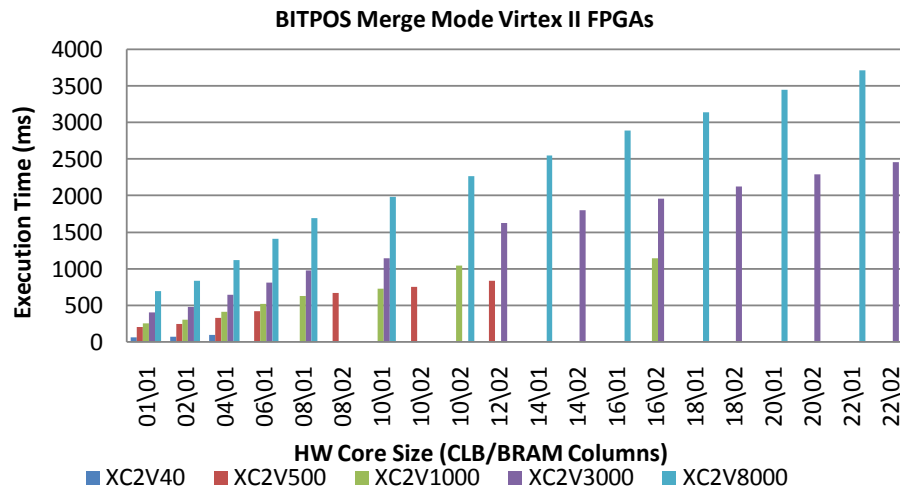


Figure 4.9: BITPOS execution time in merge mode for generating different hard cores for Virtex II FPGAs. Sizes go from 1 CLB column and use a single BRAM column (01/01 in x axis) to 22 CLB columns and one BRAM (22/01).

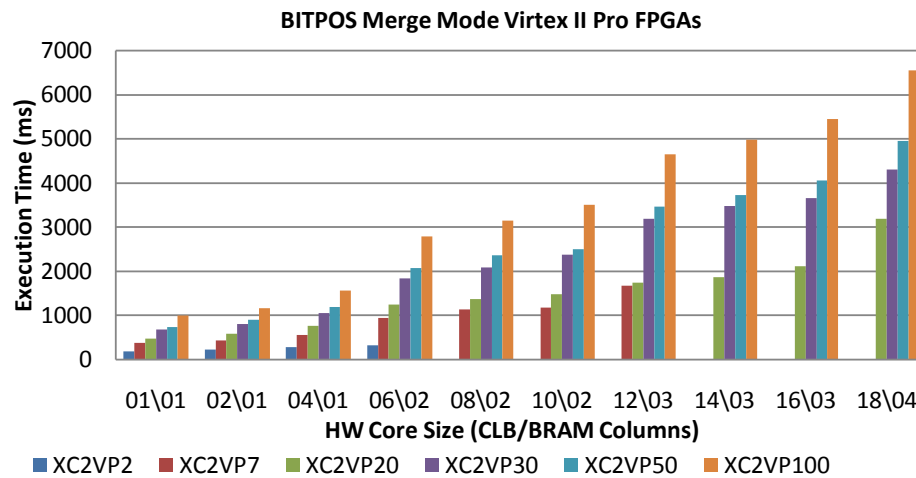


Figure 4.10: BITPOS execution time in merge mode for generating different hard cores for Virtex II Pro FPGAs. Sizes go from 1 CLB column and use a single BRAM column (01/01 in x axis) to 18 CLB columns and 4 BRAMs (18/4).

In Figure 4.11, a more detailed view of the merge mode execution time for an XC2V3000 FPGA for cores with and without BRAMs can be seen. As it has been mentioned, execution time grows proportionally with size of hard cores. From the Figure it can be noticed that the execution time for hard cores smaller than 1024 slices (7 % of the area of the benchmark FPGA) is kept under 700 ms. Nevertheless, BITPOS does not target embedded devices and, therefore, no further test will be done.

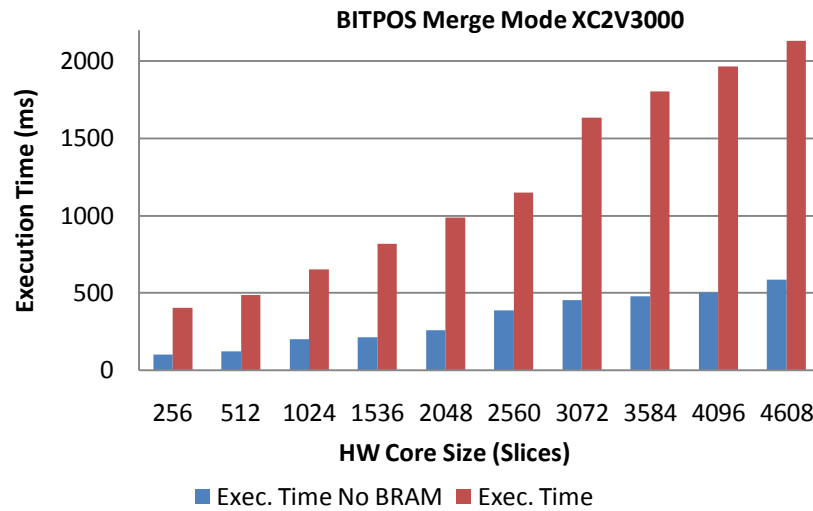


Figure 4.11: Detailed view of the execution time needed for an XC2V3000 FPGA for hard cores with different size that access different number BRAM columns and that do not access BRAM columns.

Simply as a reference, it can be mentioned that a similar tool PARBIT, needs 150 ms in slice mode to generate a pBS file with the configuration of eight CLB columns (256 slices) for a Virtex E device. BITPOS, in comparison, generates a hard core of 256 slices for an XC2V3000 device in 99 ms (see Figure 4.11). Further data of PARBIT execution time will not be included, because a fair comparison is not possible as they target different FPGAs. Additionally, the Xilinx tool BitGen needs 50 seconds to generate a hard core of 256 slices for an XC2V3000 from a native circuit description file (.ncd) using the partial mask flow (overview in the state of the art section of this Chapter).

4.8.3 pBITPOS Execution Time Results

More exhaustive timing test have been applied to pBITPOS, because execution time for this tool is important and it has to be kept as low as possible. pBITPOS characterization has been done on an embedded device, a 600 MHz XScale processor, under shared processing power conditions, with a running Linux operating system and active peripherals, like Ethernet, memory, etc.

The pBITPOS execution time in merge mode for the benchmark hard cores, previously generated with BITPOS, is presented in Figure 4.12 for Virtex II devices and in Figure 4.13 for Virtex II Pro. Core sizes cover the entire possible spectrum: from the smallest one that occupies a single CLB column and uses a single BRAM (01/01 in the figures) up to the largest possible one that occupies half of the FPGA CLB columns and one-third of the available BRAMs.

Comparing the timing results with BITPOS execution time in the same, merge mode, it can be noticed that although running on a restricted device, the pBITPOS execution time is around ten times less. This time reduction has been achieved by simplifying the relocation process. pBITPOS deals only with pBS files, where the configuration data is structured and data searching is reduced to minimum. Also, an important time reduction has resulted from the exclusion of the CRC checksum calculations.

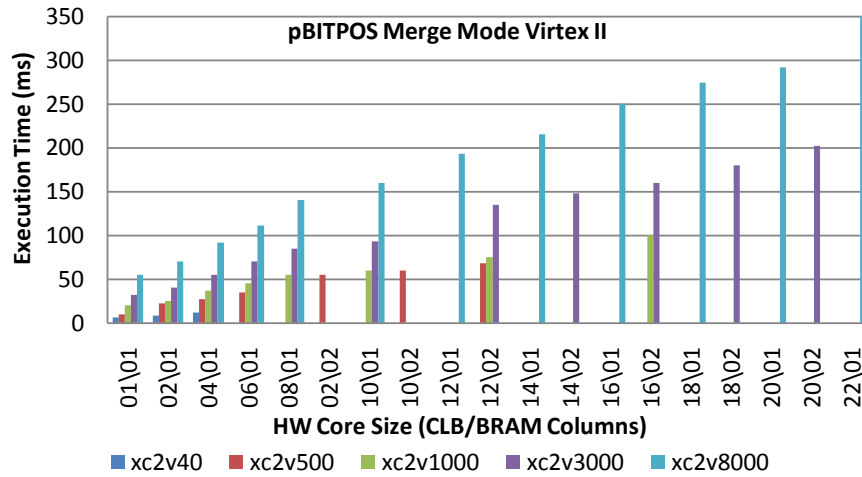


Figure 4.12: pBITPOS execution time in merge mode for Virtex II devices and for hard cores of different size. The x axis shows hard core size and number of BRAMs used by the core. Core sizes go from 1 to 22 CLB columns, while associated BRAMs go from 1 to 2.

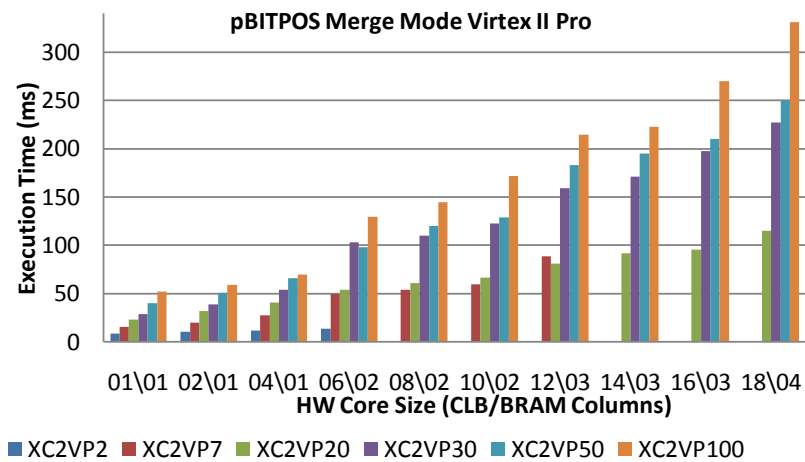


Figure 4.13: pBITPOS execution time in merge mode for Virtex II Pro devices and for hard cores of different size. The x axis shows hard core size and number of BRAMs used by the core. Core sizes go from 1 to 18 CLB columns, while associated BRAMs go from 1 to 4.

Detailed execution time results have been included for an XC2V3000 in Figure 4.14 and for an XC2VP30 FPGA in Figure 4.15. In both figures the execution time needed in merge mode for cores that access and cores that do not access on-chip BRAMs can be seen. The irregularities in the ascent steps that can be noticed in the results, related to the data of merger mode execution time with BRAMs (MM. Exec. Time with BRAMs in the Figures), correspond to an increment of the number of BRAMs that are being accessed by the hard core. From a single to two BRAMs for Virtex II (Figure 4.14) and from one to four for Virtex II Pro (Figure 4.15). To have a complete view, simple mode execution time results have been also included in the same figures.

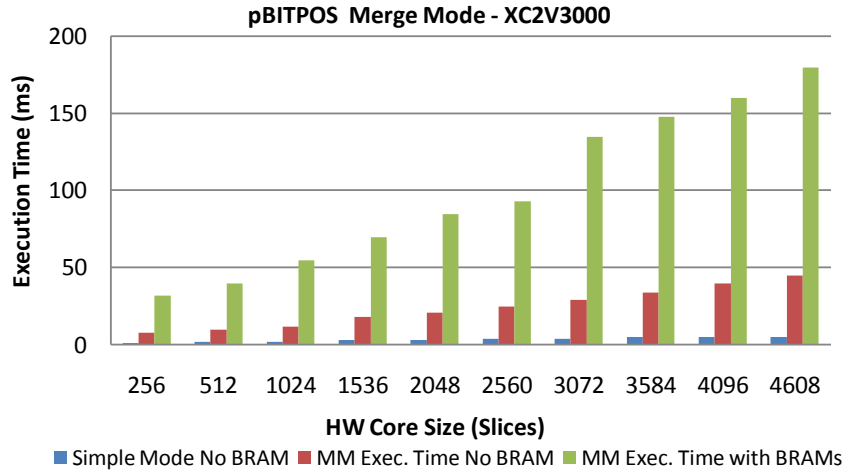


Figure 4.14: Detailed view of the pBITPOS execution time needed for a Virtex II XC2V3000 FPGA for hard cores with different sizes that access different number BRAM columns (1 or two) and that do not access BRAM columns. Additionally, the execution time in simple mode is also included.

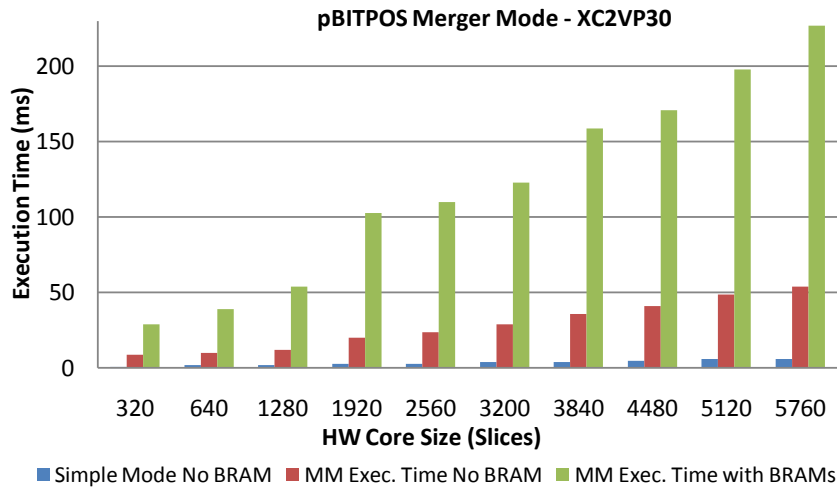


Figure 4.15: Detailed view of the pBITPOS execution time needed for a Virtex II Pro XC2VP30 FPGA for hard cores with different sizes that access different number BRAM columns (from one to four) and that do not access BRAM columns. Additionally, the execution time in simple mode is also included.

For hard cores around 1024 slices, the pBITPOS execution time in merge mode when the core accesses a single BRAM is around 60 ms. This time, like in BITPOS, can be drastically reduced to 12 ms. if the hard core does not access BRAMs. The execution time can be further reduced if only simple mode is used. In simple mode, only FAR values (a 32 bit register that defines the address of the configuration data to be loaded) are modified to do the relocation. If the hard core is generated by BITPOS (benchmark hard cores) the position of this data is well known and there is no need of parsing the entire configuration file. This results in execution times under 10 ms.

At this point, it is worthy to highlight that the Replica 2Pro filter [KP06], mentioned at the end of the state of the art in section 4.3, executes hard core relocation in simple mode (1D architectures) directly in HW (implemented in FPGA slices) almost without any delay (several clock cycles). However, on the other hand, no other bitstream manipulations can be performed, like block relocation.

The last pBITPOS execution time test is related to the, pBITPOS specific, re-target mode. As it was expected, the needed execution time in this case is closer to the execution time values related to the target FPGA. This can be noticed from figures 4.16 and 4.17, where a hard core originally designed for an XC2V500 FPGA is re-targeted to a different Virtex II or Virtex II Pro FPGAs. For example, the re-target execution time using an XC2V3000 as target FPGA, shown in Figure 4.16, is almost equal to the execution time for normal merge mode (no re-target) for the same FPGA (XC2V3000MM related data in the figure). The same can be noticed for Virtex II Pro FPGAs in Figure 4.17.

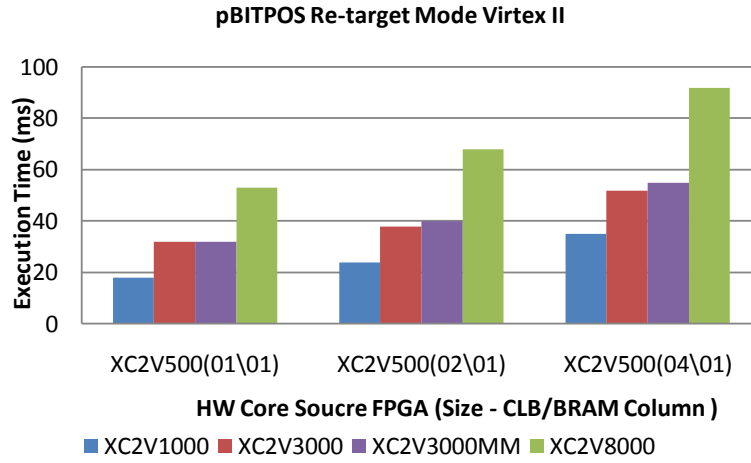


Figure 4.16: pBITPOS execution time in re-target mode for re-targeting three hard cores with different sizes, designed for a Virtex II XC2V500 FPGA, to different Virtex II FPGAs. From left to right: to an XC2V1000, an XC2V3000 and an XC2V8000. As reference the execution time in normal merger mode (no re-target) of an XC2V3000 is also included (XC2V3000MM).

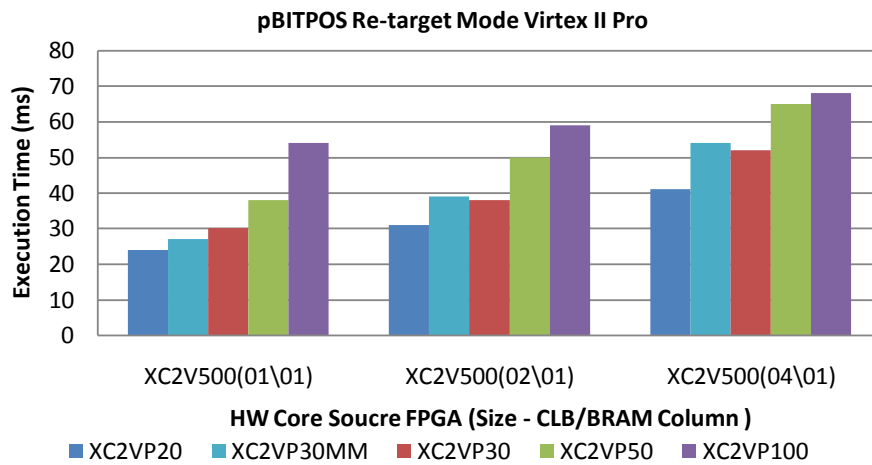


Figure 4.17: pBITPOS execution time in re-target mode for re-targeting three hard cores, designed for a Virtex II XC2V500 FPGA with different sizes to different Virtex II Pro FPGAs. From left to right: to an XC2VP20, an XC2VP30, an XC2VP50, and an XC2VP100. As reference the execution time in normal merger mode (no re-target) of an XC2V3000 is also included (XC2VP30MM).

4.8.4 Comparison with other State of the Art Solutions

To obtain an idea of the evolution in time of the state of the art, a chronology of all tools and flows described in this Chapter, along with the proposed BITPOS and pBITPOS first publication date, are included in Figure 4.18. The Figure includes, as representative academic solutions for Virtex II/Pro FPGAs, the HW relocation filter - Replica2Pro, the BitLinker tool based on JBits and Core Unifier II Pro.

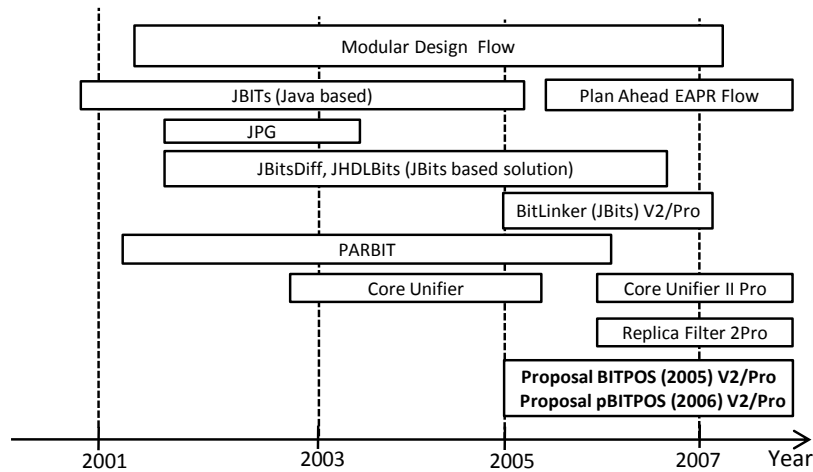


Figure 4.18: Tools and Flows for Hard Core design and bitstream manipulation time diagram.

In Table 4.7, a comparison of the features of some tools and design flows included in the related work section and the tools proposed in this Chapter can be found. For the comparison, the tools and flows are divided in the three categories distinguished in the introduction: i) design flows that end up with full or partial configuration files (Xilinx flows), ii) tools based on JBits and iii) tools based on direct bitstream manipulation. The table includes data about what type of hard cores the tool or flow can deal with (generate and/or relocate): 1D hard cores that span the entire FPGA height and/or 2D block based cores. Other FPGA elements, like embedded BRAMs available in Virtex and Virtex II based device families, are considered separate elements and therefore have a separate row in the table that shows if the tool or flow is prepared to deal with these elements (if generated and/or relocated hard cores can access these elements). Relocation capabilities are also evaluated separately. More specific features, like small bit manipulation (to work at LUT level) and FPGA re-targeting that permits to load a core in a different FPGA from the one it is designed for, are also specified. Finally, some tools/flows distinguishing characteristics are included.

From the table, it can be noticed that BITPOS shares some features with other tools (relocation and re-targeting) and adds new features (small bit and extended FPGA resources support) to provide the highest flexibility. On the other hand pBITPOS also provides the highest flexibility compared with other solutions that target embedded devices, like Replica2Pro, but as it has already be pointed out, Replica2Pro is much faster than pBITPOS.

With respect to the hard core design flow, with the Xilinx partial reconfiguration flow, an individual design has to be generated for each base and reconfigurable module combination. For instance, if there are two reconfigurable modules with two possible configurations for each one, four designs are generated. Differently, the hard core design flow proposed here has to be followed just twice (once per module configuration) or even, only once, if a virtual architecture with four slots is selected. This is possible due to the relocation capabilities of the architecture and the created tools. In the state of the art tools, the BitLinker based flow also permits to directly generate a hard core by combining several partial configuration files. However, the BitLinker philosophy is more similar to the pBITPOS one, where hard cores are merged and arranged in an embedded device in order to prepare a hard core to be loaded in the system, but it is not well suited for embedded devices. The hard core design flow proposed here is based on system templates and in commercially provided tools that facilitate the hard core design process. Designers do not need specialized knowledge, apart from the visual inspection of the design (due to the currently available place and route tools). Differently, special knowledge is required by virtual architecture templates designers.

Table 4.7: Hard cores generation and manipulation tools features summary

Parameter	Design Flows		JBits based tools		Direct Bitstream Manipulation				
	Modular Design	Plan Ahead	BitLinker	Core Unifier	Core Unifier	PARBIT	Replica2Pro	BITPOS	pBITPOS
FPGA	All		VirtexII/Pro	Virtex	Virtex	VirtexE	VirtexII/Pro	VirtexII/Pro**	VirtexII/Pro**
1D hard cores	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes
2D hard cores	V4, V5		Yes	NS	NS	Yes	no	Yes	Yes
BRAN/MULs	Yes		Yes	NS	NS	No	NS	Yes*	Yes*
Relocation	No		Yes	No	No	Yes	Yes	Yes	Yes
Small Bit	No		NS	No	No	No	NS	Yes*	Yes*
Re-target	No		NS	No	No	Yes	No	Yes	Yes
Other features	GUI		Direct runtime inter-connecting	Hard core dynamic insertion removing	There seems to be a version for Virtex II (2005-2006)		HW implementation	Hard Core design flow*	Embedded * devices
Reference	[DL04] [Xil06b]		[SF06b]	[PdMM ⁺ 02]	[MMP ⁺ 1]	[HL04]	[KP06]	[KJdITR05]	[KdITR06]

NS-Not Specified

*Thesis original contribution.

**The remaining characteristic, not marked with * are original for these FPGAs.

4.9 Conclusion

The thesis original contribution, described in this Chapter, is a complete solution for direct bitstream manipulation for Virtex II and Virtex II Pro FPGAs, that permit small, medium and coarse grain bitstream manipulation.

The solution is composed of two tools. The first one is called BITPOS and is intended to be used during the hard core design flow originally proposed in this Chapter that permits to design hard cores by non partial reconfiguration experts and without knowing the details of the target system. The second tool, called pBITPOS, targets hard core adaptation on embedded devices and has execution times that go from tens of ms. in simple mode to several tens of ms. in merge mode when the hard core uses on-chip BRAMs. The presented tools are based on direct bitstream manipulation of Virtex II/Pro FPGA configuration files. For this aim, a study of the configuration file has been performed and a complete bitstream format has been presented in section 4.5. A set of original equations have resulted from the bitstream format study and are the basics of the bitstream manipulation.

The presented tools have been characterized in terms of execution time and, depending on the target application and the embedded device where pBITPOS will be integrated, it is recommended to keep the number of BRAMs that can be accessed by a hard core to minimum. Furthermore, for dealing with such flexible architectures, execution in merge mode is mandatory, but this results in time penalties. Anyway, relocation at runtime is possible. Both tools will be used for/or integrated in reconfigurable systems in the application Chapter 5.

The tools and design flow features have been compared with other, similar tools and flows, from the related work, extensively described in the introduction part of this Chapter. From the presented data, it can be concluded that the access method, the underlying set of equations and the knowledge of the configuration file format, as well as the hard core design flow and the tools have permitted to gain flexibility and to overcome some restrictions related to the use of the technique of partial reconfiguration.

Apart from all the provided results and conclusions, probably the most important one from the work presented in this Chapter, is that the BITPOS and pBITPOS source code and executables have been provided to researchers. Several research groups have requested these tools for their own research. The last requests have been received from the Technical University of Delft (Holland), the University of New South Wales (Australia) and the Universidade Federal de Santa Catarina Mestrando (Brasil) during 2008.

Application Examples of Partial Runtime Reconfigurable Systems



While searching for the partial reconfiguration killer application, some partial runtime reconfigurable systems with different flexibility have been integrated in four application domains.

This Chapter presents *four different original applications of partial reconfiguration*. The main goal of these applications is to test several architectures with different flexibility and, to search for the partial reconfiguration "killer application", that is, the application that better demonstrates the benefits of today reconfigurable systems based on commercial FPGAs. Therefore, the presented applications are rather a proof of concept, than fully operative and closed systems.

First, a brief introduction to the partially reconfigurable systems application topic can be found in section 5.1. After that, the descriptions of the created reconfigurable systems are presented in an incremental of the architecture flexibility order. The order is as follows: i) first a wireless sensor network reconfigurable node, based on a simple architecture, is presented in section 5.2, ii) a remote reconfigurable client-server device in section 5.3, iii) an on-chip debugging system, described in section 5.4 and, iv) an on-chip communications emulation framework, based on the most complex architecture, is presented in section 5.5. Finally, conclusions related to each application can be found in section 5.6.

5.1 Introduction

Since the appearance of the first partially reconfigurable system, the benefits of the use of this technique and systems have been clear and have been mentioned in this document: silicon reuse, hot device updates after deployment, reduced design and reconfiguration times and even, reduced power consumption compared to classic full reconfigurable systems. However, these benefits are sometimes doubted and even argued. Therefore the research community has been searching for the killer application that will clearly show the benefits of partial runtime reconfigurable systems over the conventional reconfigurable ones. The pRTRS application domain is broad, from resource restricted devices running simple applications to high performance computing. Some relevant applications, from different domains are listed next:

1. Systems with adaptable coprocessor. In this context, application like encryption [Mar06], multimedia [MNT⁺04] and searching algorithms [OSM05] are widely extended. Achieved accelerations in this case are in the range of ten to several hundreds of times. In this area, reconfigurability has also been exploited for high performance computing [Gro08] and [EEAEG07].
2. DSP processing that includes all kinds of adaptive filtering. For example an adaptive FIR filter is presented in [CL07].
3. Control Systems. All types of control systems, like robots [Upe06], power control systems [VISI06], mecatronix or automotive systems control [JBHC⁺07] runtime automation control [NAS08], and even space [OMFK08] and medical [MDN⁺07] systems, simply to mention some examples.
4. Evolvable Systems. In these systems, partial reconfiguration is used to evolve circuits using genetic algorithms [Sek08]. This evolution process may also continue after circuit design leading to the concept of evolvable IP Cores. Such cores change some parameters in order to adapt themselves to new environments [Sek03].
5. Radio related applications. Software defined radio and cognitive radio [DMN⁺08] seem to be the application that has got closer to industry [DN06], [NDG05].

As it has been mentioned, several times along the document, the main reason that stops the use of partially reconfigurable systems in the industry is the lack of systems flexibility, runtime support and the complexity of the design flows and tools that required more specialized designers. This thesis work has tried to contribute in this aspect by providing design methods, tools and architectures for attenuating these problems. This Chapter presents four applications of the solutions for partial reconfiguration proposed in this thesis. The ordering, in which applications are described along the Chapter, correspond to an incremental order of the reconfigurable systems architecture flexibility.

The firstly reconfigurable system described (although it was the last to be designed) is based on a Spartan 3 FPGA and is a very simple pipeline-like, not flexible, reconfigurable system. The selected architecture for this system is similar to the ones

that can be defined with the Xilinx approach (described in Chapter 2). In this, first case, the advantages of the method presented in Chapter 2 has not been widely exploited. This simple reconfigurable system is oriented to reconfigurable wireless sensor nodes.

In the second application, a complete embedded reconfigurable client has been designed and integrated in a multimedia client-server environment (this application was developed the first). The architecture selected for this application was the 1D, bus based architecture, presented in Chapter 2 (bus-v2).

The third application is related to the use of partial reconfiguration for system debug. The reconfigurable system used in the previous application has been extended to include debug capabilities.

The last application is a complete fast on-chip communication emulation framework. It is based on the DRNoC (2D) virtual architecture, presented in Chapter 3, and is the most complex and flexible reconfigurable system.

To make the context of some applications more clear, a small introduction in the specific topic will be included at the beginning of some sections.

5.2 Reconfigurable Systems for Wireless Sensor Network

Wireless Sensor Networks (WSNs) are one of the fastest evolving and most challenging areas in today's electronic industry [ZGZ07], [Wil08].

The requirements to these resource restricted devices constantly increase. They are intended to be autonomous, low power, flexible and context aware.

In WSNs, where a spread network may be composed of hundreds or thousands of sensor nodes, one of the most complicated tasks is deployment and network support. Additionally, the increasing variety of sensors, interfaces and data processing, required to produce sensor measurements make these processes even more complex. The deployment and maintenance processes would be easier, allowing cost effective changes or adaptations, if the WSN node has higher flexibility and adaptability features.

Classic node design approaches for WSNs use a microcontroller to carry out all node tasks: node communication, sensors data processing and control, and node management [IFAC07]. Differently, new solutions based on proprietary reconfigurable systems have appeared, in order to provide higher performance and reduce power, for example [HZG06] and [SMAA06]. In the last work, an approach in which the system adapts itself (custom reconfigurable device) to execute more efficiently some predefined tasks and save energy is presented. These options, although optimized, are restricted to a specific set of application scenarios and deployments.

On the other hand, there are other solutions based on reconfigurable devices. Traditionally reconfigurable devices are not included in WSN nodes design, because of their power consumption. On the contrary, due to their flexibility and the broad new possibilities derived from reconfigurability, now some platforms include reconfigurable HW. More specifically, the Microsoft Research Labs node [DL07], includes a CPLD used for communication abstraction along the layered node platform. Also, the Tyndall National Institute has developed a modular platform [OBD⁺05], which permits to add

an FPGA layer that is mainly used for DSP computation acceleration.

The application presented in this section goes further and increases the system flexibility by designing a reconfigurable system on the node FPGA along with a simple support SW.

This work is a result of the mixture of two main research lines. One, related to modular WSN node platforms, which is not part of this thesis work and, the other one, is the topic of this thesis, partial runtime reconfigurable systems design. The WSN node platform, called *Cookie* and presented in [PdCdITR06] and [PBdCR06] has been used to create a simple remote reconfigurable system. It is also important to remark that *most of the work related to this use case is within the framework of the Master thesis "Diseño de un sistema hardware/software parcialmente reconfigurable aplicado a nodos de redes de sensores inalámbricos"* [Car09].

5.2.1 Wireless Sensor Network Node - Cookie

The *Cookie* modular platform [PdCdITR06] is composed of four main layers: processing, communications, power supply and sensors. There are several versions of almost each layer, but in the next paragraph the ones used in this application are briefly described:

1. Processing. This layer includes an 8052 microcontroller (uC) core enhanced with several peripherals from Analog Devices and a Xilinx Spartan 3 FPGA directly connected to the uC. As the FPGA does not have analog inputs, analog sensors are connected to the uC ADC (Analog Digital Converter). Differently, digital sensors are connected directly to the FPGA. This detail results in the fact that the uC to FPGA interface is different when the node deals with analog and with digital sensors, i.e. there are two uC to FPGA interfaces (analog and digital). A picture of the processing layer that holds the reconfigurable system is shown in Figure 5.1.
2. Communications: The latest version of this layer includes a ZigBee module (ETRX2 from Telegesis) that is controlled by the microcontroller through the serial UART port.
3. Power supply: The selected version is USB based, which allows power supply from a PC and provides serial programming interface for the microcontroller.
4. Sensors: The selected sensors layers is one that includes sensors for: acceleration, temperature, humidity and light.

5.2.2 Node Reconfigurability

Once the target platform has been selected, the design of the runtime reconfigurable system will be described in detail in the next subsections. Sections include a brief description of each element that is needed to create the complete reconfigurable systems: the virtual architecture, the available hard cores, the runtime software support and the reconfiguration process workflow. However, before starting, the reconfiguration scenarios are identified in the next subsection.

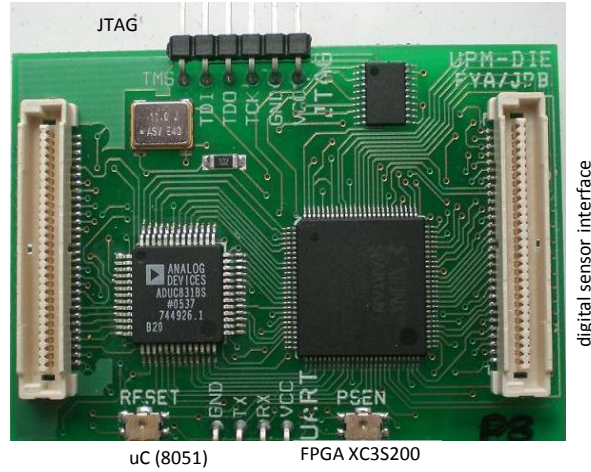


Figure 5.1: Cookie platform [PdCdITR06] processing layer.

5.2.2.1 Target Use Cases Scenarios

The dynamics of WSNs from the deployment and sensing point of view, and thus the required reconfiguration rate, are low compared to other systems. Therefore, WSNs will be considered "static". Regarding the reconfiguration process, that includes HW reconfiguration and SW reprogramming, two scenarios have been differentiated and will be targeted:

1. The first scenario includes *reconfiguration at network level* that is mostly required during deployment, where the final function of each network node is defined (this includes the used sensors and data processing) or, when the network function is changed (like in an emergency situation). Generally this scenario might cover both, SW reprogramming and HW reconfiguration. In the context of the "Cookie" platform, HW reconfiguration is mandatory when modifications affect digital sensors, as they are connected directly to the FPGA. On the contrary, SW reprogramming is required when analog sensors are involved, because they are directly connected to the microcontroller ADCs.
2. The second scenario, *reconfiguration at node level*, mainly involves HW reconfiguration. In this case, the reconfigurable array acts as a reconfigurable coprocessor where computation intensive tasks take advantage of the HW parallelism in order to lighten the microcontroller. With respect to the "Cookie" platform, when dealing with analog sensors, the entire FPGA is perceived as a coprocessor. Differently, when dealing with digital sensors, two functionalities have to be allocated in the FPGA (the coprocessor and the digital sensors control).

5.2.2.2 Virtual Architecture Design

The virtual architecture's design method general steps, presented in Chapter 2, have been followed in the next points to design the node virtual architecture.

1. First, the selected reconfiguration, in order to cover the previously described scenarios, is coarse grain. More specifically, the system has to support runtime changes in the sensor interfaces, in the data processing (coprocessor) and also in the uC-to-FPGA communication. The last one derives from the differences in the data processing of the analog and digital sensors.
2. The second step in the method is to analyze the FPGA regularity. The structure of the Spartan 3 FPGA is quite regular compared to other FPGAs. Differently from Virtex II FPGAs, this one does not have the irregularity at the center (the middle CLBs and BRAM columns). Also, these FPGAs have several BRAM columns regularly distributed along the die.
3. During the third step, the fixed area of the system has been assigned to be the leftmost and rightmost FPGA sides, as it can be noticed from Figure 5.2, where a schematic view of the virtual architecture is presented. The right one is used to access the node sensors, while the left fixed area is in charge of the communication with the external microcontroller. The remaining portion of the FPGA, defined as reconfigurable area, has a very regular structure and is divided into reconfigurable slots. This area in the target FPGA (XC3S200) spans throughout 16 CLB columns.

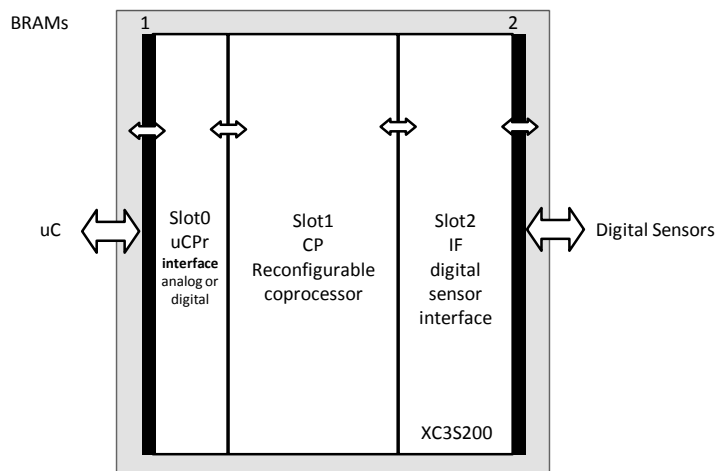


Figure 5.2: Spartan 3 FPGA virtual architecture. The architecture is composed of three slots with different size and specific functionality. The FPGA communicates on the left side with the Cookie uC and on the right side with the Cookie digital sensors. The on-chip communication is based on CLB neighboring connection macros.

4. Along the thesis, a solution for Virtex II slots distribution in 2D, where slots had equal size has been presented. This solution has not been applied to the

Spartan 3 FPGA, because in the available 16 CLB columns of the reconfigurable area, at least three slots had to be allocated in order to cover both scenarios: one for digital sensor control, one for data processing (coprocessor) and one for the communication with the microprocessor (analog or digital). Furthermore, it has been estimated that more than 150 CLBs (at least 8 full-height CLB columns in the target FPGA) will be required for the data processing slot. Furthermore, the system has been considered static and each slot has a specific, defined, function that might not change. As a result, a 1D *non relocatable slot distribution* with three slots composed by CLBs (following the slot definition of Chapter 2), but with different sizes and with specific function, has been defined. Nevertheless, in the future a more flexible architecture could be designed (if the functions of the WSN node are enhanced and a bigger Spartan 3 is available) and pBITPOS can be easily extended to support these FPGAs. The bitstream format of the Spartan 3 FPGA is very similar to the Virtex II one, and thus the SW adaptation is direct (software parameters for Spartan 3 FPGAs have to be added).

5. For defining the on-chip communication during the last method step, as relocation is not foreseen and the application is pipeline based, the simplest solution, based on point to point neighboring links has been selected. For the interconnection implementation, as Spartan 3 FPGAs do not include TBUF, CLB based macro structures were the only choice. Some of them have been taken from [Xil06a] and others, bidirectional, have been designed with the FPGA Editor.

A complete view of the designed virtual architecture can be seen in Figure 5.2. The architecture is composed of three slots which width is directly related to the hard cores to be held: i) slot0, on the left side, is 2 CLB columns wide (a total size of 48 CLBs and 128 slices) and allocates hard cores that define the communication with the uC (uCPr), ii) slot1 is 8 CLB columns wide (192 CLBs and 768 slices) and is used to allocate the uC coprocessor (CP) and iii) slot2 on the right side is 6 CLB columns wide (96 CLBs, 384 slices) and holds the digital sensors interfaces (IF). The connection of each slot of the architecture to the neighboring one through CLB Macros can also be seen in Figure 5.2.

5.2.2.3 Reconfigurability Control Software

A simple control SW, which runs on the node uC, has been created with the goal of evaluating the possibility of sending new configurations using the WSNs restricted network. The node control receives or sends new SW programs and HW configurations (hard cores) through the network and loads them in the microcontroller program memory. Regarding the FPGA programming, the available Spartan 3 FPGA does not have an internal configuration port (ICAP). The remaining FPGA configuration options are Select Map and JTAG.

To keep the modularity and generality of the Cookie platform, JTAG has been selected as the reconfiguration interface. Furthermore, a SW implementation of the JTAG controller [inc07], provided by Xilinx, has been used and runs on the uC. The controller uses Boundary Scan configuration files as input and controls the FPGA JTAG interface signals connected to a microprocessor port. The input for the FPGA programmer is simply a pointer to the configuration to be loaded and therefore it satisfies the

modularity property of the platform. On the other hand, the main disadvantage of using JTAG is the higher reconfiguration time, compared to other programming interfaces.

The available in the Cookie platform ZigBee communication module has been used for network data transmissions. The uC of the node, used to transmit the configuration data, sends commands to the ZigBee module to manage the communications channel and transmits the raw configuration data.

5.2.2.4 Platform Hard Cores

The hard cores currently available to be loaded into the reconfigurable system slots are:

1. uC protocol (uCPr) hard cores to be loaded in slot0. There is one version for accessing digital sensors and one for analog data treatment. item
2. Coprocessors (CP) hard cores to be loaded in slot1. An eight bit moving average filter that constantly collects measured data and gives an average value as a result and a FIR filter. Sensor Interface (IF) hard cores to be loaded in slot2. There are HW configurations available for the temperature and the accelerometer sensors in two versions: one uses the on-chip multipliers (MULs) that are next to slot2 (see Figure 5.2) and a second version that implements the multiplications in LUTs.

For some slots, there are also empty and feedthrough configurations. The first ones are simple, empty pieces of the FPGA, while in the second one, slot input are connected to slot output macros. The last is suitable for instance, when the reconfigurable coprocessor is not used and slot2 needs to be feedthrough.

5.2.2.5 Reconfiguration Process Work Flow

The work flow that has been followed for the reconfiguration process is presented in Figure 5.3. This process, like the hard core design flow presented in Chapter 4, starts by selecting the virtual architecture to be used (architecture definition files). After that, the conventional (ISE) design flow can be used. This flow stops with the generation of a full configuration net-list file (.ncd), from which hard cores are generated using the BitGen "PartialMask" Flow (described in section 4.2.1.).

Another approach is to use the Xilinx partial reconfiguration design flow, based on ISE and the PlanAhead tool (described in section 4.2.1). Differently from the first approach, the second one directly results in partial configuration files.

Independently from the method used to generate partial configuration files, the next step is to use the iMPACT tool to generate the binary formatted boundary scan configuration file (.xsvf), needed to partially reconfigure the node FPGA. The last steps (see Figure 5.3) are to send the new configurations to the Cookie uC that is in charge of the last step, which is to reconfigure the FPGA.

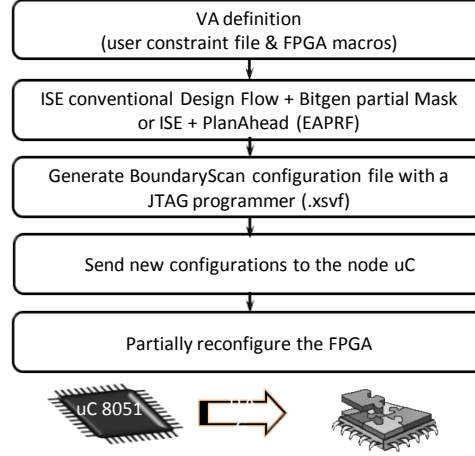


Figure 5.3: Reconfiguration Process Work Flow Diagram.

5.2.3 Reconfiguration Evaluation Parameters

In the previous sections, the pRTSR for the WSN node has been designed and the partial reconfiguration work flow defined. Now, a set of parameters have been defined in order to evaluate: i) if a reconfiguration is worthy to be carried out and ii) the delivery of new HW configurations and SW programs through the WSNs network. The evaluation parameters, related to the main embedded device restrictions, memory and energy, are based on relative values of cost.

The first group of parameters are related to energy and its definition as the power consumed during the time needed for finishing a certain task. It has been distinguished:

1. **Transmission Energy.** The energy spent during HW configurations and SW programs transmission. The combination of a SW and a HW configuration (hard core) is referred as node configuration.

$$E_{trans} = T_{trans} * P_{trmode} \quad (5.1)$$

Where T_{trans} is the time needed to transmit a configuration, and P_{trmode} is the power consumption in transmission mode.

The same reasoning is applied to the following parameters.

2. **Reception Energy** is the energy needed to receive and save a new node configuration.

$$E_{rec} = T_{rec} * P_{recmode} \quad (5.2)$$

3. **Retransmission Energy.** This energy differs from the sum of the reception and transmission energy (it is lower) as it might not require access to external memory.

$$E_{ret} = T_{ret} * P_{retmode} \quad (5.3)$$

4. **Reconfiguration Energy** is the energy needed to successfully complete a single node reconfiguration.

$$E_{reconf} = T_{reconf} * P_{reconfmode}. \quad (5.4)$$

5. **Standby Energy** is the energy spent in standby mode

$$E_{sbmode} = T_{reconf} * P_{sbmode}. \quad (5.5)$$

The cost of doing a task is measured as the part of the total currently available node energy (E_{node}) that has to be consumed to finish that task:

1. **Transmission Cost**

$$C_{trans} = \frac{E_{trans}}{E_{node}} \quad (5.6)$$

2. **Reception Cost**

$$C_{rec} = \frac{E_{rec}}{E_{node}} \quad (5.7)$$

3. **Retransmission Cost**

$$C_{ret} = \frac{E_{ret}}{E_{node}} \quad (5.8)$$

4. **Reconfiguration and total reconfiguration costs** have been already defined in section 3.5.3. Here, the relation related to power restricted devices will be used and is included next for completeness:

$$C_{reconf} = \frac{E_{reconf}}{E_{available}}, \text{ where } E_{available} \text{ in this case is } E_{node}. \quad (5.9)$$

$$C_{totalreconf} = \sum_{i=1}^{N_{reconf}} C_{reconfi} \quad (5.10)$$

Apart from the energy, in this resource restricted device, the memory is also an important issue. In this background, memory related cost parameters have been defined. The **cost of having a configuration (Clib) in the local library** is defined as the memory portion of the total available memory in the node (N_{mem}) used by the configuration (CN_{mem}) minus a coefficient related to its importance CN_{imp} :

$$C_{lib} = \frac{N_{mem}}{CN_{mem}} - CN_{imp}, \text{ where } 0 > CN_{imp} < 1 \quad (5.11)$$

Additionally, the **Total Remote Reconfiguration Cost (CTRR)** is defined as the sum of all the C_{trans} , C_{rec} and C_{ret} needed to reach the Node Under Reconfiguration (NUR), where N_{hop} is the number of hops. Notice that this value does not include the total device reconfiguration cost as both processes are not directly related.

$$C_{trr} = C_{trans0} + \sum_{i=1}^{N_{hop}} C_{ret} + C_{rec}N_{hop} + 1, \quad (5.12)$$

Generally, a certain task is considered as worthy to be done by a node only when the task cost, minus a coefficient related to the importance of the task, is less than a defined cost threshold (Ctrsh). Importance coefficients related to both, memory and energy, are derived locally, on each node, depending on the WSNs main objective that is defined by a centralized administrator in charge of sorting the network priorities. Additionally, the non total costs values are between 0 and 1. Thus, a task cannot be done by the node if its cost is higher than 1.

5.2.4 Test and Results

The parameters presented in the previous section have been applied to the selected Cookie platform in the following subsections and also, the transmission of new configurations through the network is evaluated.

5.2.4.1 Transmission, Reception and Retransmission Data Rates

For testing configurations transmission/reception and retransmission, a node configuration, which corresponds to slot2 (4 CLB columns wide slot) and the FPGA programming SW (SW_FPGA_Prog) has been transmitted to a Node Under Reconfiguration (NUR) using different types of transmission media: i) using the available wireless communication in the sensor node and ii) using a serial cable connection. For the data transmission and retransmission, two different packet formats have been taken into account and can be noticed in Table 5.1: one of 16 bytes and another of 8 bytes. Results show that the overhead in the transmission time when using a wireless media is around 2.5 times more, and the data rate is almost three times less, compared with a cable download.

Table 5.1 summarizes transmission/reception (both values are equal) time and data rates for the testing configuration and for both transmission media (file sizes can be found on Table 5.5 and Table 5.6). Differently, retransmission measures, based on the same SW and HW configuration, are as follow: the time needed for retransmitting the hard core is 100,15 for 8 bytes packets and 70,65 for 16 bytes packets. The time needed for retransmitting the mentioned SW configuration is 30,21 sec. for 8 bytes packets and 20,48 for 16 bytes packets.

Table 5.1: Data transmission time and rate for the JTAG configuration SW, and for a hard core that corresponds to slot2 - 4 columns wide.

Mode	SW (sec.)	HW(sec.)	Total (sec.)	data rate (Bytes/s)
Cable 8B	19,06	73,5	87,22	333,04
ZigBee 8B	49,11	190,22	219,31	131,95
Cable 16B	13,72	53,56	67,28	470,70
ZigBee 16B	29,09	114,75	143,75	220,30

5.2.4.2 Cookie Cost Parameters

In order to calculate cost parameters, the node available energy has to be known. This value can be calculated dynamically by constantly subtracting the node consumption from the initially available energy. There is a solution to dynamically measure the node power consumption, where the Cookie node uC is in charge of calculating it, but it has not been integrated so far in the reconfigurable system.

The Cookie platform, in its current version (not power optimized), uses two AA batteries of 1Ah at 1.5 V. The power, consumed by the node in standby mode (P_{sbmode}) is 60 mW, 150 mW in reconfiguration ($P_{preconfmode}$) and sensing mode ($P_{sensemode}$) (the uC manages indistinguishing both processes), 210 mW in radio transmission (P_{trmode}) and reception modes (P_{rcmode}) and 110 mW in retransmission mode ($P_{retmode}$). If the node is supposed to last for at least 100 hours (this time is based on the normal, non partial reconfiguration, function of the node), then it has a limit energy budget of 0,01Ah. For the tests, it has been supposed that the one hour period has just started and then, the available energy is $E_{node} = 30 \text{ mWh} = 1800 \text{ mWmin}$.

5.2.4.3 Transmission/Retransmission Energy and Cost

Transmission energies and costs have been calculated using equations (5.1) and (5.6), and can be found in Table 5.2 for both packet standards. Furthermore, retransmission energies and cost parameters, calculated with equations (5.3) and (5.8), can be found in Table 5.3.

Table 5.2: Transmission/reception (both values are equal) energy and cost parameters.

Mode	Etran (mWmin)	Ctrans
SW ZigBee 8B	171,88	0,095
SW ZigBee 16B	101,82	0,056
HW ZigBee 8B	665,77	0,369
HW ZigBee 16B	401,63	0,223

Table 5.3: Retransmission energy and cost parameters

Mode	Eret (mWmin)	Cret
SW ZigBee 8B	55,38	0,030
SW ZigBee 16B	37,54	0,020
HW ZigBee 8B	183,60	0,102
HW ZigBee 16B	129,89	0,072

Taking into account the presented transmission and retransmission data, the resulting total remote reconfiguration cost (equation 5.12) for a long distance connection, with one

hop, for the most costly transmission (8 bytes transmission of the hard core) is $C_{trr} = 0.84$ ($C_{trans} + C_{ret} + C_{rec}$). This value is directly related to the total energy to be consumed by the three nodes involved in the device update that, in this case, is almost equivalent to the entire energy budget of a single node.

If a distributed control is defined, each node will decide if the given task is worthy, depending on the partial costs and the task importance. On the other hand, if a centralized control is used, then total costs will be taken into account and the final decision will rely on a centralized controller.

5.2.4.4 FPGA Configuration and Memory Cost

FPGA partial configuration file sizes (generated using the work flow presented in the previous section) in both .bit and .xsvf format (the one transmitted in the network) are presented in Table 5.4 for hard cores that use single slots and hard cores that use a slot (slot2) and a BRAM/MUL column.

Table 5.4: Hard cores memory cost

HW Configuration	.bit (KB)	.xsvf (KB)	Clib
Slot0 (2 CLBs)	8,7	8,9	0,13
Slot1 (8 CLBs)	32,9	33,3	0,52
Slot2 (6 CLBs)	24,9	25,2	0,393
Slot2 + 1 BRAM/MUL column	52,3	54,5	0,851

Concerning SW programs file sizes, an accelerometer control interface (SW_ACCIF), that includes some data computation in the microcontroller, and the FPGA programmer software (SW_FPGA_Prog), have been used as an example in Table 5.5.

Table 5.5: SW programs memory cost

SW Programs	.hex (KB)	Clib
SW_ACCIF	4,4	0,06
SW_FPGA_Prog	6,4	0,1

The memory used for keeping hard cores and SW programs is the microcontroller program memory. Taking into account that this memory is 62 KB, Clib has been calculated with equation (5.11) for hard cores in Table 5.4 and for SW programs in Table 5.5, assuming that CNimp for all configuration types is the lowest possible value ($CN_{imp} = 0$). As reference, it is important to mention that a complete XC3S200 configuration file is 131 KB and has an unaffordable memory cost ($\gg 1$). This demonstrated an advantage of using partially reconfigurable systems. Anyhow, it can be also noticed that the memory cost of hard core files is much higher than a SW one.

Reconfiguration costs can be found in Table 5.6 and have been calculated with equation (5.9) and the measured reconfiguration time, also included in the table. From the

presented data in tables 5.6, 5.2 and 5.3, it can be noticed that the cost of transmitting a configuration is much higher than the reconfiguration cost. Usually for a context switch, several hard cores are needed to be loaded in the systems. As the transmission cost is very high, it is important to appropriately manage the local node memory and to keep hard cores as small as possible. The table also includes the reconfiguration cost for reconfiguring the entire FPGA that is almost ten times more (another advantage of partial reconfiguration).

Table 5.6: Reconfiguration cost Parameters

HW Configuration	RecTime(sec)	Ereconf	Creconf
Slot0 (2 CLBs)	3,6	9	0,0050
Slot1 (8 CLBs)	7,8	19,5	0,0010
Slot2 (6 CLBs)	4,3	10,75	0,0059
Full FPGA(estimated)	15	37,5	0,0208

5.2.4.5 Hard Cores Area Requirements

The FPGA area needed by some hard cores is presented in Table 5.7: i) for uC interfaces (slot0), only the digital version is included (uCDPr) as the analog does not consume FPGA LUTs (it is a feedthrough hard core), ii) for coprocessors (slot1), a moving average filter (HW_AVRF) is included and ii) for the digital sensor interfaces (slot3), one Freq/Period temperature sensor (HW_DSTMP) and one PWM based accelerometer (HW_DSACC). Notice that there are two versions of each digital sensor interface hard core: one that uses the FPGA embedded multiplier (_v2) and one that does not use them (_v1). Access to the embedded BRAM/MUL columns is guaranteed for hard cores that occupy slot0 and slot2 (see Figure 5.2). The table also includes the usage percentage of the corresponding slot. From the table it can be noticed that all hard cores occupy less than 60 % of the available slot area, therefore no problems related wires that cross slot borders have been noticed.

Table 5.7: Hard Cores area requirements

HW Configuration	Used MULs	Used LUTs	Used FFs	Used Slices	% Slot Slices
uCPr (slot0)	0	2	18	9	4,6
HW_AVRF (slot1)	0	253	306	273	35,5
HW_DSACC_v1	0	170	68	311	54
HW_DSTMP_v1	0	71	40	127	22
HW_DSACC_v2	2	127	68	232	40
HW_DSTMP_v2	2	63	40	111	19

5.2.4.6 Reconfigurable System Validation

Using the mentioned hard cores, two applications have been defined as use case, where a context switch from an analog sensors control (AS_HW_AVRF) to a digital sensor (DS_HW_AVRF_TMPIF) is performed. The two application setups are listed next.

1. The first application is composed of one analog sensor and the FPGA is used as a coprocessor where average filtering is applied (AS_HW_AVRF). For building it, in slot0, a feedthrough configuration is needed, in slot1 the AVR filter hard core and finally, slot2 is left empty. The mapping of the application in the reconfigurable system (FPGA Editor screen shot) can be seen on the left side of Figure 5.4.
2. A digital temperature sensor with average filtering (DS_HW_AVRF_TMPIF). For building it, in slot0, a uCPr configuration is needed, in slots1 the AVR filter hard core and finally the temperature sensor interface has to be loaded in slot2. The mapping of this application in the partially reconfigurable system can be seen on the right side of Figure 5.4.

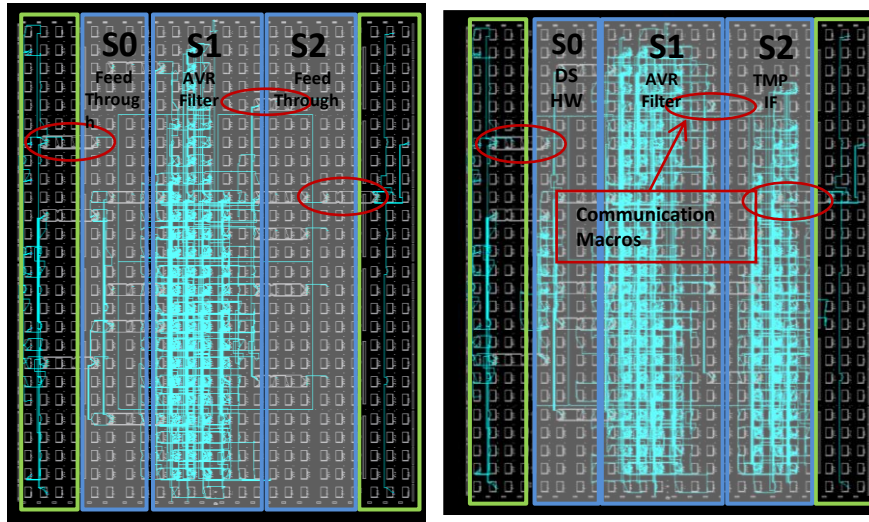


Figure 5.4: System use case application FPGA Editor screen-shots. Analog sensors node configuration (AS_HW_AVRF) on the left side and the digital sensors configuration example (DS_HW_AVRF_TMPIF) on the right side

The reconfiguration sequence, employed for validating the context switch is as follows:

1. Initial full FPGA configuration required for the virtual architecture definition. This configuration, sometimes, might contain hard cores allocated into slots.
2. Send through the network the feedthrough configuration for slot0 and the AVR filter hard core to reconfigure slot1 and run the AS_HW_AVRF application. The total reconfiguration cost of this process is $C_{totalreconf} = 0,0158$.

3. Send uCPr and HW_DSTMP_v1 hard cores and configure the FPGA to pass to the DS_HW_AVRF.TMPIF application with a total cost of $C_{totalreconf} = 0,0109$.

After this, for instance a different digital sensor interface, like the HW_DSACC in any version can be loaded in slot2 with a smaller cost of $C_{totalreconf} = 0,005$.

5.2.4.7 Results Evaluation

Generally, the efforts of adding the partial runtime reconfigurable system in the node have not been high. Furthermore, the cost of programming the FPGA is negligible compared to the total node power consumption. The highest penalty (highest cost) is the configuration transmission and reception.

However, reliability in WSN applications is an important aspect, even more if the network is also used for transmitting device updates. Tests have shown that the 16 bytes transmission fails with a high rate in long distance connections (with intermediate hops) compared to 8 bytes based transmissions that are much more reliable, but more costly, reaching values of 0,369 (for HW Zigbee 8B in Table 5.2) for hard cores transmission. That is, almost 40 % of the available energy budget has to be invested on this task. The ZigBee standard is not well suited for this kind of remote updates and therefore the amount of data to be transmitted has to be kept as low as possible and partially reconfigurable systems in this aspect are better than the classic reconfigurable ones.

To summarize, the penalty of exploiting the platform flexibility by transmitting new configurations through the network is reduced by using a partial runtime reconfigurable system.

Currently, this work is being improved by designing an intelligent reconfiguration control system that, based on the defined cost parameters, will take distributed and/or centralized decisions. Also, a special Cookie memory layer is being designed to be able to keep more node configurations locally.

5.3 Remote Reconfigurable System: ENAMORADO

The application presented in this section, similarly to the previous one is oriented to remote reconfigurable devices. But, differently from the WSN application, here the complexity of the entire systems is considerable. In this application, flexibility and versatility are the main goals. Furthermore, here, the final application is not restricted, like in the WSN node where sensing was the main point.

In the following subsections an entire content and configurations delivery environment is presented. *This work is enclosed in an IST project called, "Enabling Nomadic Agents in a Multimedia ORiented Architecture of Distributed Objects" (ENAMORADO), partially funded by the European Union.*

First, the overall ENAMORADO environment, shown in Figure 5.5, is briefly reviewed and each part is described. Afterwards, *the focus is put on the contribution of this thesis work and one the main challenges in the project, that is, the design and integration of the reconfigurable client device.*

5.3.1 ENAMORADO Environment

The overall system, shown in Figure 5.5, is comprised of various elements with different functionality, resulting in a scalable and easily extensible client-server environment. Each element of the environment has been developed independently by the ENAMORADO consortium partner with clear definitions of the interfaces among them.

An IP-based network infrastructure where end-users can be connected through a number of different access networks, either wired or wireless (e.g. UMTS, WLAN) is supported. The mechanisms, provided by ENAMORADO, based on the notion of content and device adaptation techniques and adjust the communication profile in client-server environments. The proposed mechanism uses the real-time transmission protocol/real-time control transmission protocol (RTP/RTCP) for the content delivery. The environment covers all the elements of the client-server multimedia oriented system; from the live event production center to the reconfigurable platform (see Figure 5.5). All the components will be briefly described next, but the emphasis has to be put on the ones directly related to the client reconfigurability: the interactive server and the network measurement and simulation.

1. The production center holds the *Content Production Environment (CPE)*, which is primarily responsible for the production of the provided contents, in different formats, taking either live material and/or pre-recorded information. It offers a set of real-time production tools, which allow gathering live material from a number of sources, enhancing it with meta-data that reveals information about the content (title, keywords, duration, format, and version), and producing multimedia files like video-clips, images and audio files. The produced contents can be either directly retransmitted to the client device or held in a content repository from where they are pulled on demand.
2. The *Interactive Multimedia Server (IMS)*. The CPE interacts with the IMS module,

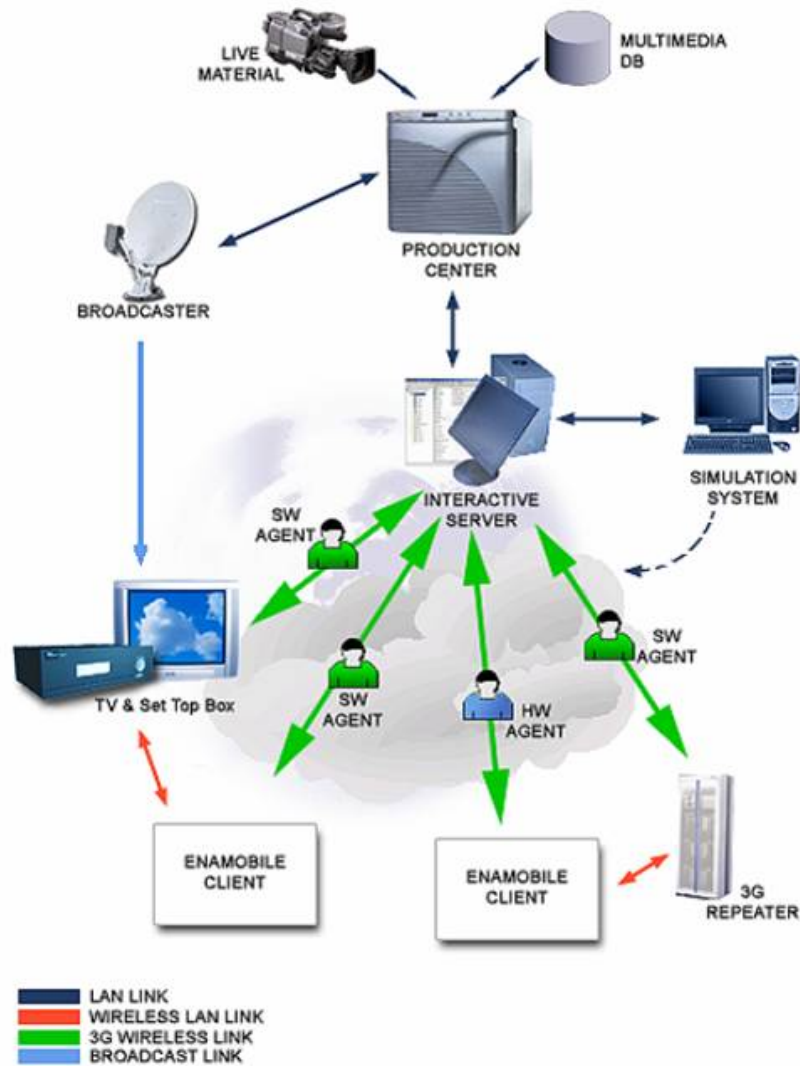


Figure 5.5: ENAMORADO (Enabling Nomadic Agents in a Multimedia ORiented Architecture of Distributed Objects) environment general view.

which constitutes a central regulator of the system operability. The IMS supports the modeling of the full delivery scenario, from the production center to the terminals, as it controls the content delivery functionality in various formats and towards several devices. It also initiates the decision management process and retrieves content and event notifications from the CPE. The IMS accepts and

analyses user requests for content delivery in various protocols (HTTP, WAP, etc.). It is the responsible for analyzing the profile data, send by the client devices, and take the final decision of which is the most appropriate configurations to be delivered to the client and, also to notify the Streaming Server to provide the client with content in the appropriate format.

3. **Simulation Server (SS).** In order to provide the best communication policy, the IMS consults the SS which selects the most appropriate content format depending on both user requirements and network conditions. The Simulation Server encompasses part of the system intelligence, as it provides an efficient approach for the optimum adjustment of the communication profile and adoption of the reconfiguration strategy, according to the user preferences, device capabilities and the network conditions. The SS analyzes current network conditions based on both, real-time monitoring results and off-line simulations in order to determine the most appropriate content. The SS communicates with the Streaming Server, which monitors RTCP messages to obtain session statistics, combines them with direct network measurement and provides an estimation of the network status in terms of congestion and quality.
4. The ENAMORADO environment deals with *reconfigurable clients, called Enamobiles*, able not only to update its software by downloading "software plug-ins", but also to adapt its hardware. This allows filling the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than fixed hardware. New hardware and software configurations are transparently downloaded from the IMS to the client and installed in the respective embedded microprocessor and FPGA for a proper content reproduction. The FPGA can allocate more than one configuration at the same time and it is possible that some of these configurations need to be replaced without interrupting the mission of the other tasks. This implies the development of a dynamic partial reconfiguration strategy. A full set of tools to control, negotiate, download, replace, delete and change configurations position within the FPGA have been developed and will be described in the next subsections.

5.3.2 ENAMORADO Reconfigurable Client

One of the main innovations in the context of ENAMORADO is the introduction of reconfigurable clients in the system. Differently from other client devices, reconfigurable clients are able not only to update their software by downloading "software plug-ins", but also to adapt their hardware. New hardware and software configurations are selected and downloaded from the IMS to the client and installed in the respective embedded microprocessor and FPGA in a fully automate way. Differently from the previous use case, the needed flexibility and versatility by the reconfigurable system are higher as the foreseen applications to be executed in the system are broad (audio, video, control, data treatment, etc).

The inclusion of a flexible pRTRS in the client device permits to have a parallel system where different applications can coexist in the same FPGA making simultaneous use

of the FPGA resources. The FPGA might not be dedicated to a single application like in the previous use case where sensing was targeted. The basis of the system flexibility has been defined by the reconfigurable system virtual architecture and the hard core's management in the own embedded device. System flexibility has to be also supported by other ENAMORADO environment components (like the IMS), as it will be demonstrated in the next points.

5.3.2.1 Virtual Architecture

The selected virtual architecture for this use case is the XC2V3000 1D, bus based system, Bus-v2, shown in Figure 5.6 and presented in Chapter 2. For completeness, a summary of the system properties can be found next:

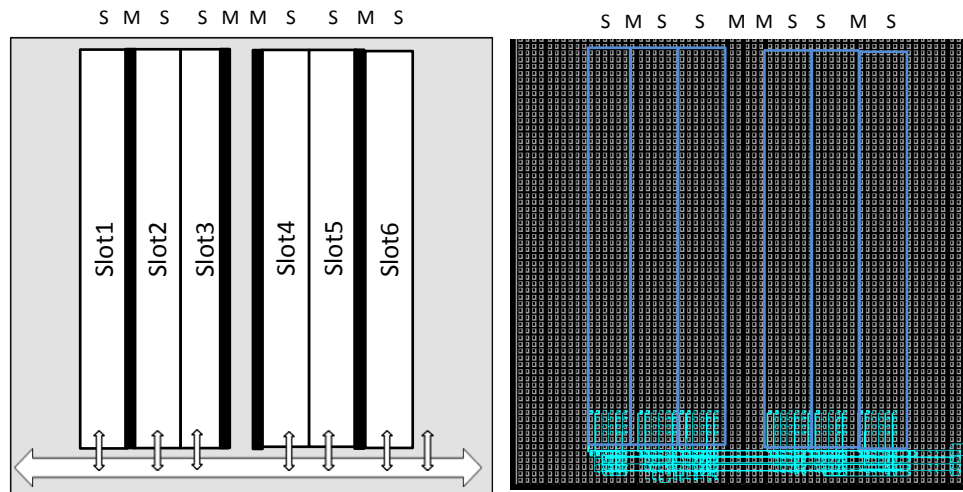


Figure 5.6: 1D bus architecture for the target XC2V3000 FPGA (bus-v2), along with the related FPGA architecture definition string in the upper part.

1. The fixed area spans the rightmost, leftmost and middle CLB columns.
2. The reconfigurable area is divided into 8 slots of 6 CLBs width. But the rightmost and left most slots are additionally assigned to the fixed area to control the microprocessor to FPGA data exchange and the FPGA programming, resulting in 6 slots available to load hard cores.
3. The on-chip bus communication is composed of a total of 50 wires. Sixteen are for bidirectional data transmission, 16 for unidirectional, from slots to the fixed area, and 18 are unidirectional from slots to the fixed area.

As it was highlighted in Chapter 2, the architecture permits a hard core to be freely allocated in any slot position and also several positions are possible for hard cores that

need access to on-chip BRAMs. It permits, as well, slots to be grouped to allocate bigger hard cores. These characteristics make the architecture suitable for this application, where hard core shapes and functionality is not known in advance.

To design hard cores for this system, the hard core design flow presented in Chapter 4 has been followed. The resulting hard cores, along with a hard core definition are combined to create hardware configurations that are uploaded into the IMS and included in its HW configurations repository. The use of the proposed flow has permitted external designers, with no partial reconfiguration knowledge, to design hard cores and create HW configuration.

5.3.2.2 Client Device Reconfiguration Management

To deal with the FPGA reconfigurability, a middleware (FPGA manager) has been defined and has been integrated in the system software. This middleware is composed of a set of applications for managing, deleting, shifting and loading hard cores into the FPGA.

For making the system versatile and portable, a hardware abstraction layer that specifies the virtual architecture currently available in the reconfigurable system, has been defined. This information is stored in a file, which contains the definition of each slot borders, accounted in FPGA columns, and a simple string that defines the slot utilization. In the string, slots are marked with "S" (upper case) when they are unused and with "s" when they are used. BRAM/MUL columns are marked with "M" and "m" respectively. For example, the bus-v2 architecture, illustrated in Figure 5.6, has the string "SMSSMMSSMS" shown in the upper part of the Figure. This string is updated each time a new hard core is loaded in the FPGA. This simple solution, provides versatility to the system. If a new virtual architecture is defined, simply the architecture definition file has to be updated. Furthermore, each hard core has a similar string that is included in its hard core definition file (extended description file). For instance, if a core is composed of two slots and a BRAM, its hard core definition string would be "SSM".

The FPGA manager general view can be seen in Figure 5.7 and each element, is described next in the order of its execution call.

1. **FPGA Allocator.** Once a new hard core has been received in the device, as there is not a specific place for each hard core (like in the WSN case), first the allocator has to check if a hard core fits into the system. For this aim, the FPGA Allocator performs a search of the best allocation place. The search is based on comparing data extracted from the core definition file, the FPGA current state coding string and the preloaded hard core status. Hard core active state is defined in an FPGA registry file. The allocator checks this data to select a candidate to be removed, just in case there is slot high occupation. The Allocator applies a greedy set of rules for hard core positioning preferences, like overwriting the first inactive core (if needed) and filling the FPGA from left to right, if empty.

After the Allocator, pBITPOS is called, if reallocation is needed.

2. **pBITPOS.** pBITPOS, presented in detail in Chapter 4, has been slightly adapted to fit in the presented FPGA manager. The lighter version, with the lowest

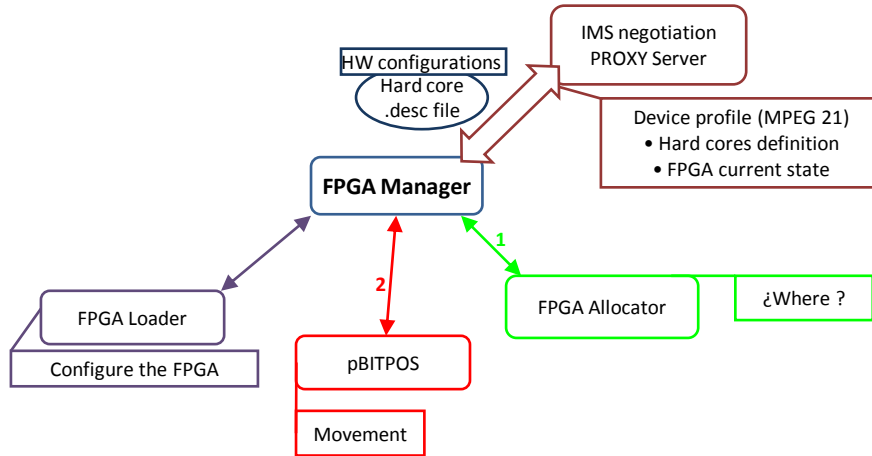


Figure 5.7: FPGA reconfiguration control middleware.

execution time (no CRC calculation), has been used. Also, instead of using a reference to the description file location as input, all needed parameters are given directly in the command line. The tool parameters are calculated on the embedded device, depending on the results given by the FPGAs Allocator and the virtual architecture definition.

The virtual architecture configuration bitstream is also needed during the hard core relocation process as pBTIPOS is executed in merger mode. This bitstream is the one that is loaded by default in the FPGA (initial full FPGA configuration) and contains the slots borders definition and the on-chip communication infrastructure.

3. **FPGA Loader.** The last tool to be called is the FPGA loader, that is a HW/SW application in charge of the FPGA reconfiguration using the ICAP port. The SW part prepares the configuration data that is sent to the FPGA through a shared memory between the FPGA and the external microprocessor. The HW part, allocated in the virtual architecture fixed area, controls the on-chip ICAP port.

The manager is also in charge of loading empty and dummy hard cores in the respective slots when needed (if a hard core has been inactive for a long time) and also, to control and update the FPGA state string and the FPGA registry file. The registry file defines each core loaded in the FPGA. It has been kept as simple as possible and is a text file composed of several fields: hard core ID, slot position, used memory, a field that shows if the core is active and a time reference.

Apart from this, the manager is in charge of the communication with the upper layer of the SW stack that includes a proxy server responsible of the configuration negotiation, with the IMS server, process, described in the next point (see Figure 5.7).

5.3.2.3 Reconfiguration Client-Server flow

New configurations are selected by the IMS server taking into account a set of device characteristics (device profile). Part of the included data is related to the device reconfigurability, like the type of reconfigurable device (FPGA) and its architecture. The file also contains a list of cores, available locally in the unit, either currently loaded, or kept in memory. The profile is formatted in an XML file and is locally updated, in the client device, each time a new core is installed on it or a device characteristic is changed (for instance the amount of memory). The profile is sent to the IMS server each time a user accesses the service and it is then used during the configuration selection process. The device profile has been defined by extending the MPEG 21 part 7 "Digital Item Adaptation" [MPE] profile to include fields related to device reconfigurability characteristics. The fields related to the device reconfigurability (codec's or configurations hard cores) are listed next, and a piece of such pseudo file can be found, as an example, in Figure 5.8:

```
<!-- ##### -->
<!-- Definition of Reconfigurable System -->
<!-- ##### -->
<complexType name="ReconfigurableDeviceType">
  <complexContent>
    <extension base="dia:DeviceClassCharacteristicsBaseType">
      <attribute name="Type" type="string" use="optional" />
      <attribute name="Vendor" type="string" use="optional" />
      <attribute name="Model" type="string" use="optional" />
      <attribute name="VirtualArchitecture" type="string" use="optional" />
    </extension>
  </complexContent>
</complexType>
```

Figure 5.8: Device profile - reconfigurable system related fields. The shown fields are part of the extension applied to the MPEG21 standard in order to provide reconfigurability support.

- Configuration or codec related fields
 - EnaCodecID - Identifies a specific hard core in the ENAMORADO environment.
 - Vendor - Indicates who has designed the codec.
 - FPGASize - Specifies the hard core size in used FPGA columns and rows.
 - HwInterface - Describes communication interfaces, used when the hard core is installed in the FPGA.
 - EstimatedPowerConsumption - Indicates the power consumption of a configuration. In special power conditions the IMS could deliver lower power configuration.

- Device reconfigurable system related fields
 - ReconfigurableDevice - Describes the reconfigurable device (custom ASIC, commercial FPGA) of the terminal.
 - Type - Describes the reconfigurable device (FPGA, CPLD, etc) of the terminal.
 - Vendor - The name of the vendor (Xilinx, Altera, Atmel, etc).
 - Model - The model of the device (i.e. XC2V3000).
 - VirtualArchitecture - Indicates the number of slots and their distribution (1D or 2D).
- Device current state related fields
 - FreeHWCapabilities - Describes the free resources inside the FPGA that can be used for loading new hard cores (number of free slots).

With all these specified profile fields and the definition files, the complete transparent flow used for configurations negotiation between the client device proxy server and the IMS is described next:

1. The user and service client logs into the service through a web page using the reconfigurable device.
2. The device sends to the IMS the recently updated device profile file.
3. The client browses through the provided services in the web and when any content is selected to be displayed in the device (image, video, audio, etc), the IMS analyses the device profile that is automatically sent by the client device, and the held in the IMS client profile that defines the client preferences and services contracted. The result of this analysis is a set of contents along with the respective configurations, if they are not already available in the client device, prepared to be sent to the client device.
4. After that, the IMS consults the simulation server to check the state of the used communication link and gives as result a recommendation of which is the best data rate to be used.
5. With this data, the IMS finally selects the best content and configuration that are sent to the client device.
6. This new configuration is received in the client device by the proxy server that calls the FPGA manager.
7. After the device update process, the content is downloaded from the repository of the content production environment.

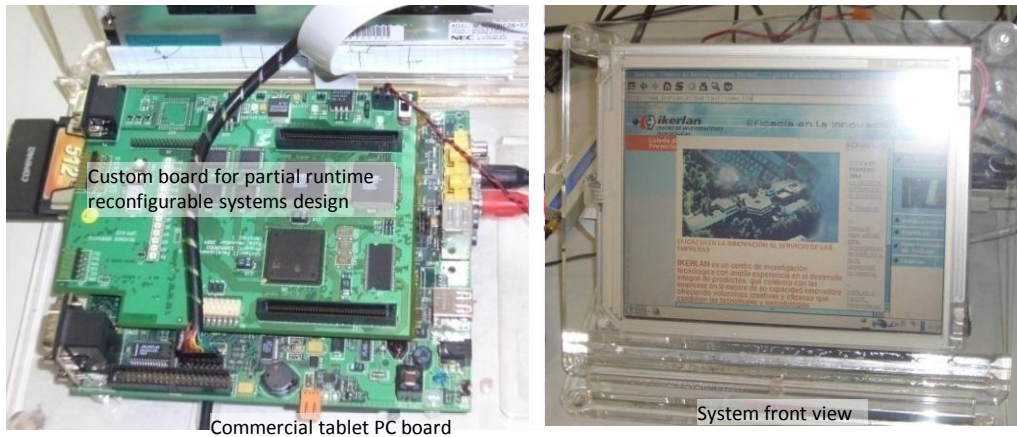


Figure 5.9: ENAMORADO reconfigurable client device composed of a commercial tablet PC platform (XINGU) with an XScale processor, shown on the right side, and a custom board specially designed for partially reconfigurable systems prototype shown on the left side.

5.3.3 Tests and Results

The reconfigurable client demonstrator and development platform, shown in Figure 5.9, is composed of:

1. A commercial tablet PC development board, called XINGU, with a 600 MHz XScale microprocessor and two wireless networks, Wi-Fi for the short range connectivity and GPRS or 3G for the wide range. The board also has a compact flash memory and an expansion connector that is connected to the system bus. The system front view can be seen on the right side of Figure 5.9. The XScale runs a Linux operating system with the OPIE environment and the entire client software stack that includes; the low level drivers to control the FPGA programming, the FPGA manager previously described and the proxy server for the communication with the remote server.
2. A custom board with a Virtex II XC2V3000, specially designed to meet partial reconfiguration requirements (peripheral elements placement among others) can be seen on the left side of Figure 5.9. The board also includes two dual port, high speed, banks of memories for data transfer with an external microprocessor and a CPLD in case external slot connection is required. Some FPGA top border IOs are connected to the CPLD device IOs. The CPLD is used as a programmable switch to interconnect slots, similar to the CPLD included in the ESM (Erlangen Slot Machine) platform described in section 2.3.6.

5.3.3.1 Reconfigurable System Evaluation

The selected bus-v2 virtual architecture has been previously, individually, validated in Chapter 2 where some relocation and partial dynamic reconfiguration tests have been performed. Here, the tests described next target the entire reconfigurable system validation. The tests have been divided in three groups that cover different system elements.

First, some benchmark hard cores from those used in Chapter 4 have been selected to test the FPGA manager. Selected hard cores sizes go from single slots "S" to several slots along with a BRAM "SSM". The time needed to execute the FPGA Manager (all the tools that it includes) goes from half a seconds in the best case (S slot configuration and no relocation), to one second for the biggest benchmark hard core (SSM) and with relocation.

The hard core size influence in this time is negligible as their size affects only the pBITPOS execution time (deeply studied in Chapter 4) that for the used benchmark hard cores is below 200 ms.

The most time consuming operation is the FPGA programming process that takes 90 ms for an "S" core. Compared with the normal ICAP configuration time that is in the ns range, this time is extremely high. As it has been mentioned, the FPGA loader is a HW/SW application, where the FPGA receives the programming data from the microprocessor and controls the internal configuration port. The FPGA to microprocessor communication is done through the tablet PC development board expansion port that is shared with all the peripherals available in it (wireless communications and memory). As result, data transactions with the FPGA are the non-optimized step of the process and the easiest to improve, if a dedicated interface is reserved for FPGA communication.

With this first test, the FPGA manager flow has been validated using dummy cores that have no functionality.

5.3.3.2 Reconfigurable Client-Server Flow Evaluation

In the second test, the system partial reconfiguration flow is validated. The basic tests performed in Chapter 2 have been repeated increasing the hard cores complexity. In this case accumulators, multipliers and buffer based hard cores have been designed using the hard core design flow presented in Chapter 4.

The flow was followed by a person without partial reconfiguration knowledge that, starting from the virtual architecture definition and template files, has designed different MAC hard cores of "S" and "SM" type that exchange data with the external microprocessor. For example, a core that takes some data from a memory, performs some MAC operations and sends the results back to the outside of the FPGA using the on-chip bus. These cores, along with each hard core definition file were uploaded to the IMS server. Afterwards, the entire reconfiguration flow, from the user logging to the FPGA programming, including file updates, was tested. By repeating the process several times in a specific order, the FPGA manager was forced to relocate hard cores and to update the FPGA architecture state string, while the proxy server was forced to

update the device profile and the FPGA registry files.

Test results have shown that the entire process takes around 15 seconds in the worst case (executing all the flow steps), measured from the time right after the content selection by the user to the end of the FPGA programming. It is important to mention that the time needed to download the content depends on the network state. Therefore, the time needed from the reception of a new HW/SW configuration to the end of the system update process was measured, and it was around 6 seconds.

5.3.3.3 Overall ENAMORADO System Evaluation

After the reconfigurable system and the flow were validated, further tests were done with bigger hard cores that are part of complete device configuration or codec's (HW/SW). Two codecs were designed and integrated in the ENAMORADO environment:

1. A filter that takes an 8x8 pixel portion of and JPEG file from the microprocessor, saves it in a BRAM, applies a filter to modify the image and returns data to the microprocessor to be displayed. This process is repeated for the entire image. The hard core occupies three slots and a BRAM column - "SSM" and has been fully integrated in the reconfigurable system. After the core has been installed in the system, the user can apply the filter to any image downloaded from the content server. Again, this hard core was designed by an external person and following the design flow of Chapter 4.
2. An open source software video decoding application has been selected, the VLC (Video LAN Client media player), and the source code has been modified. Some resource consuming functions, like among others the IDCT (Inverse Discrete Cosine Transform), which is the main decompression operation for many multimedia formats, has been rewritten and adapted to be a hard core. The main problem here was that this hard core occupies almost all reconfigurable slots. Therefore for this huge hard core, the configuration string is the entire FPGA reconfigurable area "SMSSMMSSMS" and thus it was not fully integrated in the reconfiguration flow. In this case the FPGA was updated under full configurations.

Generally, the tests performed have shown that the reconfiguration flow of the system permits to have higher flexibility and versatility compared with the previous use case. The hard cores that can be consumed by the systems are broad in terms of used slots and can be designed by designers without detailed knowledge of partial reconfiguration.

The limit is put by the available space like was in the case of the VLC player. Also, problems with the on-chip communication have been noticed. Flexibility in this sense is restricted to the on-chip communication protocol that is defined in the FPGA fixed area and each hard core that is loaded in a slot has to be able to support the defined communication protocol. To overcome this restriction and improve the on-chip communication flexibility, the DRNoC architecture has been proposed in Chapter 3 and is the base for the application presented in section 5.5.

5.4 Reconfigurable Systems Debug

This section deals with the use of partial reconfiguration techniques to insert different debug and monitoring modules in a reconfigurable system. The use of this technique has some advantages: first, the debug module can be replaced by another one without stopping or modifying the execution of the core under debug, and second, the technique can also be applied for systems running in normal operation mode, where the behavior of a newly added core, and its interaction with the rest of the system needs, to be verified prior to be integrated in it.

The proposed, basic debug and monitoring system, is shown in Figure 5.10. The reconfigurable system is connected through a JTAG cable to a host PC, where, *a tool for debug and monitoring FPGA based systems, called CHDT (Configuration Handling and Debug Tool) [GdlTAR04], is running. The CHDT tool has been created at the CEI lab as a result of a research work [GdlTAR04] and it has been successfully used for debugging different HW systems in the last years.* Therefore it has been selected for studying the possibility of using partial reconfiguration for debugging reconfigurable systems.

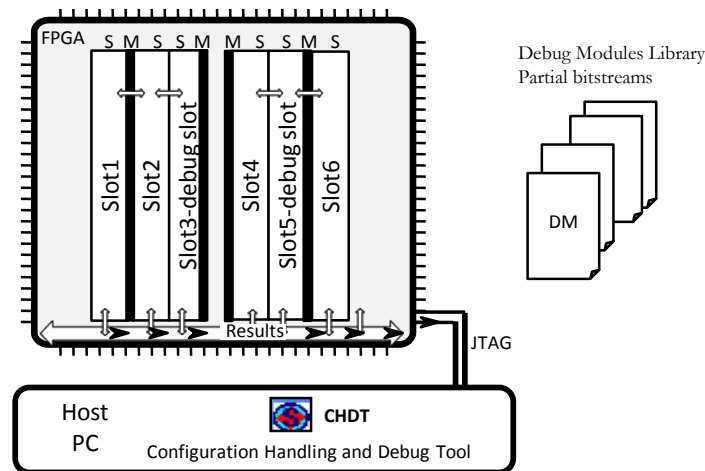


Figure 5.10: Using partial reconfiguration for debug.

5.4.1 Debug System

In order to enable debug, based on partial reconfiguration, horizontal macro structures have to be added to the selected virtual architecture. In this case the target architecture is the bus-v2 used in the remote reconfigurable system described in the previous section. The added, to the bus-v2 architecture, macros that connect two neighboring slots can be seen on Figure 5.10, where the modified architecture is shown. In the same figure, two debug slots, where debug modules are loaded, are marked as an example. Debug and monitoring modules (DM) are hard cores that are loaded into the FPGA in

an adjacent slot position to where a Core Under Test (CUT) is loaded. DMs collect and/or control the CUT test outputs/inputs that are connected through the debug port. Following the hard core design flow, and using the modified virtual architecture definition files of the enhanced bus-v2 architecture, DMs hard cores were designed. All the designed DM cores have been kept in a local library and loaded when needed.

All data retrieved from the DM(s) is displayed graphically with CHDT, or with custom interfaces which are specific for each module. The tool has also the possibility to detect via JTAG, which debug modules are currently loaded in the system. This is done reading the JTAG UserCode register. Unfortunately this CHDT feature is not applicable when dealing with partial reconfiguration, because the reconfiguration of the UserCode register can affect the communication of running modules with peripheral devices, connected to the right band FPGA IOBs. The JTAG controller is allocated in the top right FPGA corner upon the right band IOBs.

The application of this debug methodology has some implications in the design of the CUT. Signals that are going to be monitored and/or controlled by the monitoring system have to be routed to the debug port (the horizontal macro). This stage is performed using CHDT. Using the CHDT GUI, the user selects the signals to be extracted from any point in the design hierarchy, and the tool connects them to a generic debug module. CHDT automatically modifies the core hardware description files, adds the debug modules from a library and connects the ports of both modules. All these modifications are done in the lower hierarchy levels of the design and do not affect the virtual architecture template file (the highest hierarchy level).

The debugging system can be divided into three parts allocated in different FPGA areas. First, the CUT, which can be positioned in any slot. Second, the DM, which can be placed in a neighboring slot, right or left to the CUT. The communication between both modules is performed using the debug port. The third element is the JTAG controller that is allocated in the fixed FPGA area (FPGA rightmost slot). The controller is in charge of modifying and retrieving data from debug modules, and to send it to the CHDT tool running on the host PC.

Communication between the DM and the JTAG controller needs to be performed through debug-reserved wires in the on-chip bus communication structure. There are some wires left in the on-chip communication infrastructure, after the bus protocol implementation, which are used by the JTAG. The JTAG communication requires ten wires for accessing and controlling the JTAG accessible user-defined registers, which are inside the DMs.

In order to include debug capabilities in the reconfigurable system, like the used in the previous application described in section 5.3, the control system has to be forced to reserve a slot that is adjacent to a debug-ready CUT. A simple way to do this is to specify that the core to be loaded in the system will require two slots. For instance, following the slot definition string specification included in section 5.3.2, the debug version of a hard core of "MS" type is a CUT of type "MSS", where the second S represents the slot where the debug module will be allocated. The "MSS" core can be handled by the proxy server and the FPGA manager just like any other core, and once it is placed in whichever slot position, any DM can be loaded in the reserved slot from the library. Once the module functionality has been validated it can be changed by the real hard core.

CHDT contains a library of debug modules which range from very simple modules like event detectors, monitor registers or event counters, to complex modules like in-circuit emulators for microprocessor SW debugging or trace modules (which would use the embedded RAM of the FPGA for data storage).

5.4.2 Tests and Results

As an application use case, simple hard core DM modules have been created to test the proper operation of some state machines and MAC hard cores that occupy one slot ("S" type cores) from the ones used in the previous application (section 5.3). CUTs were loaded in slot2 (see Figure 5.10) and different DMs were loaded in slot3 using partial reconfiguration without affecting the behavior of some running modules allocated in slot1 and slot5. The CUT outputs were properly monitored with the CHDT tool shown in Figure 5.11.

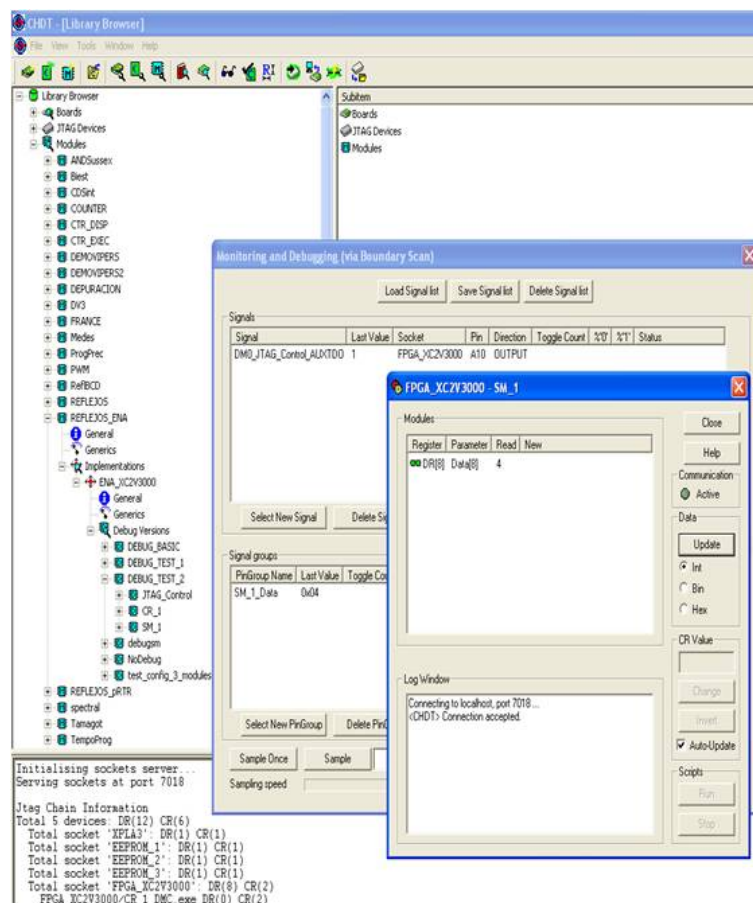


Figure 5.11: CHDT execution screen shot while dealing with a hard core debug module.

It was checked that the CUT works properly by searching for a certain, expected, transitions and register values, by using a pattern detector as DM module. Currently, there are four DMs in the hard core library.

It is extremely difficult to debug a reconfigurable system where hard cores are constantly swapped and the access to internal signals is restricted. In this aspect, we have found that the on-chip debug process was improved due to the use of partial reconfiguration in combination with CHDT. Debug modules were operated from specific module controller windows, as in the regular use of CHDT for non-reconfigurable systems. However, some manual operations were required in order to place the monitoring signals in the appropriate positions for connecting them to the debug port bus macro.

5.5 On-chip Communications Emulation Framework: DRNoC EMUL

The selected application in section 5.3 was a remote reconfigurable client. The final, general conclusion, from that work was that more flexibility was needed in the communication. Therefore a new approach for on-chip communication, called DRNoC, has been proposed in Chapter 3.

In the embedded device, the decision of what to configure (and when) was taken by the remote server that holds the biggest part of the intelligence. On the other hand, due to the versatility of the defined client system, it is not difficult to extend it to support the DRNoC architecture. For this purpose, the device definition related files have to be modified on the client side and core administration on the server side has to be improved. In that case, from the remote server, along with the downloading of new slots configuration: new cores for slots/core, new network interfaces for RNIs and, new routers for RRM's have to be downloaded.

Differently, after an analysis of NoC prototyping state of the art (presented afterwards), it has been decided to explore the use of the most flexible and complex architecture from the ones presented in this thesis as a base for an original *on-chip communications emulation framework, based on a partial reconfiguration*, where data analysis is performed in a host PC and the selection of the best configuration is done by a user.

NoCs have rapidly evolved during the last years and a lot of research efforts have been oriented to NoC based SoCs design, as well as prototype approaches. One of the main challenges in NoC design is to find the optimal NoC solution for a given application. Several methods and design flows that permit to perform design space exploration, at different abstraction levels have been proposed. Higher abstraction levels permit to rapidly evaluate different mapping and NoC implementation options without paying the time cost of long simulations. For instance in [MVM06], a framework for MPSoC NoC systems modeling, simulation and evaluation, based on System C models is presented. There, systems are generated matching application and platform models. Other frameworks are based on object-oriented languages, like the presented in [CCG⁺04], where a C++ library is built on top of SystemC, or based on the Matlab simulation environment, like [MA07]. Other approaches generate NoC topologies getting the application graph representations or application descriptions as starting point, using analytical and/or heuristic methods. Examples of these are SUNMAP [MM04] that, based on the Xpipes [JMBM04] NoC generator, creates topologies modeled in System C, or in [JFBCR⁺08], where systems are specified in XML. These approaches are very suitable for system design early stages as they permit to have the fastest design space exploration. There are other HDL or mixed (System C and HDL) solutions for NoC modeling, like MAIA [OMP⁺], where NoC parameters are defined by a user and NoCGeN [CP04] where the NoC topology can be selected. Lower level, VHDL or RTL, permit to have accurate results, but are more time consuming. Usually, at this abstraction level, traffic generators and traffic consumer models that simulate real Core behavior are used in order to reduce the simulation time.

On the other side, there are FPGA based emulation solutions proposed to drastically reduce the simulation and therefore systems evaluation time. For instance, in

[GAM⁺05], a HW-SW FPGA based emulation framework is presented and combined with the Xpipes environment. Four orders of magnitude of speedup are reported in that work. Emulation, depending on the FPGA available area, may permit to test NoCs using real applications instead of traffic models. In [OML⁺07], four real applications mapped into a NoC and prototyped in an FPGA are presented.

Nevertheless, the main disadvantages of emulation based solutions are:

1. Synthesis time: every time a system parameter needs to be changed, the system has to be re-synthesized, re-placed and re-routed (from here after, the processes of synthesis, place and route will be referred simply as synthesis). This is not a real problem if a system has to be synthesized once, but if the goal is to come up with the optimal system implementation, several combinations have to be synthesized and emulated. An approach to overcome the synthesis problem is to have several system versions implemented in the FPGA and switch between them. This forces cores used during the emulation process to be different from the one used in a real system, as they have to include more functionalities, and this results in increased area overhead.
2. Extraction of measured data from the FPGA: the FPGA has much more limited resources, in this sense, in comparison with a SW based simulation.
3. The available FPGA area permits only to emulate relatively small systems. In [WHS07], a solution for this problem is proposed. There, sequentially, parts of a parallel system are loaded into an FPGA and emulated. Thus, the emulation process is separated in several stages that have to be concluded before having a complete system emulated. Although that, speedups of 80 to 300 in comparison with System C simulation are reported. Anyhow, each new FPGA generation has more logic available and thus permits to emulate bigger systems.

From this short state of the art, it can be noticed that fast solutions for the prototyping are still being investigated and, that FPGAs are gaining importance in NoC based SoCs design. Therefore, emulation was the selected application for the DRNoC architecture, and the focus has been mainly put on attenuating the synthesis time disadvantages.

It has been considered that the DRNoC architecture could be a good solution for building a fast emulation framework, as the architecture permits to exploit partial reconfigurations and thus re-synthesizing of the entire SoC is not needed.

The main, original, idea is to build the entire system to be emulated with reusable hard cores by loading them on the DRNoC architecture using partial runtime reconfiguration. The second idea is to demonstrate the advantages of using pRTRS, specially DRNoC, in the emulation domain.

The DRNoC architecture, like the bus based used for the remote client application (section 5.3), permits hard cores to be reallocated along the slot architecture (vertically and horizontally) and to be grouped to allocate bigger cores. But DRNoC, also permits to modify the on-chip communications and this makes the systems the most complex and flexible from the all those presented in this thesis. Apart from the synthesis, emphasis has also been put on measured data extraction and configurations administration. The last of the before mentioned drawbacks is more related to FPGAs

technology and FPGA integration, therefore it is out of the scope of this thesis. Related to this, it is true that the biggest, real, DRNoC that has been implemented so far is a 2x2, and this highly restricts the systems that can be emulated. However, the 2x2 DRNoC will be used to validate the idea of applying partial reconfiguration in emulation. Anyway, a bigger 4x4 DRNoC has been mapped in the biggest FPGA of the Virtex II family. Furthermore, the emulation system is prepared to work with other DRNoC mappings, and can also be adapted to support other FPGA families.

The emulation work flow and the entire emulation framework is presented in the next subsections.

5.5.1 Work Flow - Design Space Exploration

A general view of the work flow for design space exploration, based on partial reconfiguration is presented in Figure 5.12. In some aspects it is similar to other NoC design space exploration interactive flows, like [GADMB07] and [MH06], but here hard core re-usability is exploited and also, the system is not restricted to NoC communications schemes.

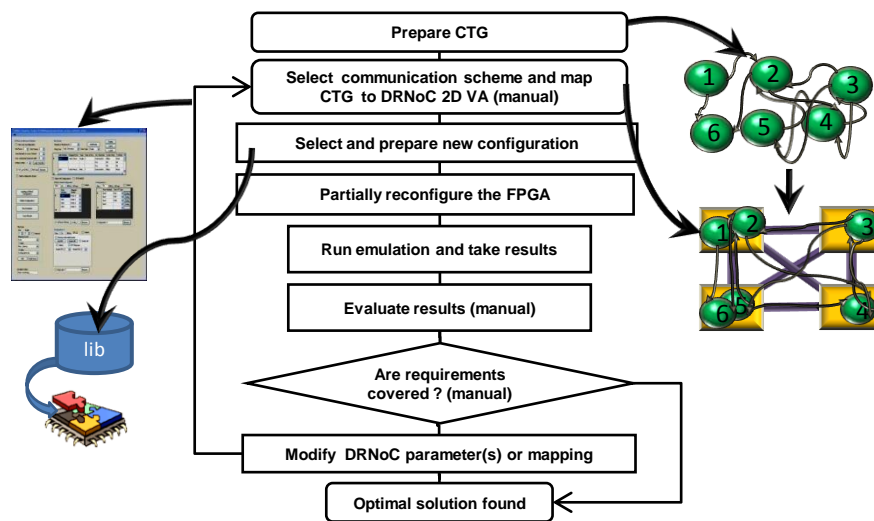


Figure 5.12: DRNoC design space exploration work flow. The main, distinguished characteristic of the flow is that it is based on hard cores re-usability. Notice, that some steps of the flow are manual.

The main distinguishing characteristic of the flow is that, the communication schemes to be emulated are entirely built by assembling reusable hard cores, from the hard core library, by partial reconfiguration operations.

The flow starts with a mapped application Communication Task Graph (CTG), where application tasks are assigned to system cores. For each node of the CTG, a suitable DRNoC element model, traffic receivers (TR) and/or traffic generators (TGs), are

assigned if they are available in the hard core library, or generated if they are not available (using the hard core design flow). Afterwards, in the first step of the interactive flow, the new CTG is mapped to DRNoC architecture, currently available in the emulation system (it is a 2x2 DRNoC). TGs and TRs are assigned to slot/cores, and the communication scheme is defined. NIs, routers, P2P and P2M links are assigned to RNIs and RRM. *From here after, each CTG to DRNoC mapping, along with an assigned communication scheme will be referred as a configuration.* Also, in this step, measurement points are defined in terms of number and tracked TGs nodes (measurement points, defined in section 3.4.3.4, are composed of a set of registers that hold packet statistics and are allocated in receiving nodes). After all configurations to be emulated have been setup, emulation is executed. For each DRNoC configuration, FPGA bitstreams, available in the hard core library or manually provided to the emulation system, are arranged and loaded in the FPGA. All configurations are emulated consecutively, and results related to each measurement point are independently saved. Later, results can be plotted, analyzed by a user and, if needed, new DRNoC configurations can be added with modifying the CTG to DRNoC mapping and/or the communication schemes definition. This process continues until the best communication scheme and CTG mapping has been found.

5.5.2 DRNoC Emulation Framework

A general view of the emulation framework is presented in Figure 5.13. The framework is distributed in three platforms: i) The DRNoC FPGA board, where measurement points are allocated in TRs and receiving nodes NIs, ii) the measurement system HW control mapped on an XUP board with an XC2VP30 FPGA and iii) a host PC that runs the DRNoC emulation SW. It has been necessary to allocate the HW control in a separate platform in order to have an entire FPGA for the DRNoC. The XUP board has been selected because building the control system on the Virtex II Pro PPC is relatively fast, permits easy updates and in the future it could be included in the DRNoC FPGA itself.

The DRNoC approach and the virtual architecture design have been extensively presented in Chapter 3, here, the HW control system is described in this subsection, while the DRNoC emulation SW is the subject of the next subsection.

The HW control system main element is a custom peripheral that is connected to the PPC on one side and directly to the DRNoC FPGA on the other side. The custom peripheral is in charge of controlling the emulation process (run, stop, reconfigure and continue), pulling data out from measurement points registers and buffering them.

Following [GIP⁺07], the system measurement points are allocated in TRs and in the associated TRs NIs. Data is extracted from these measurement points using an AMBA APB bus interface that shares the same FPGA area with the Xmesh communication channels (Xmesh, defined in section 3.4.1, is a mesh with diagonal links). There is one AMBA APB bus for each DRNoC row. As the PPC system runs at 100 MHz and the DRNoC FPGA runs at 25 MHz, an asynchronous four phase communication interface has been used for connecting both systems and a bridge has been included on the DRNoC FPGA fixed area to multiplex the internal APB buses.

In the current HW control implementation, fourteen lines have been separated for

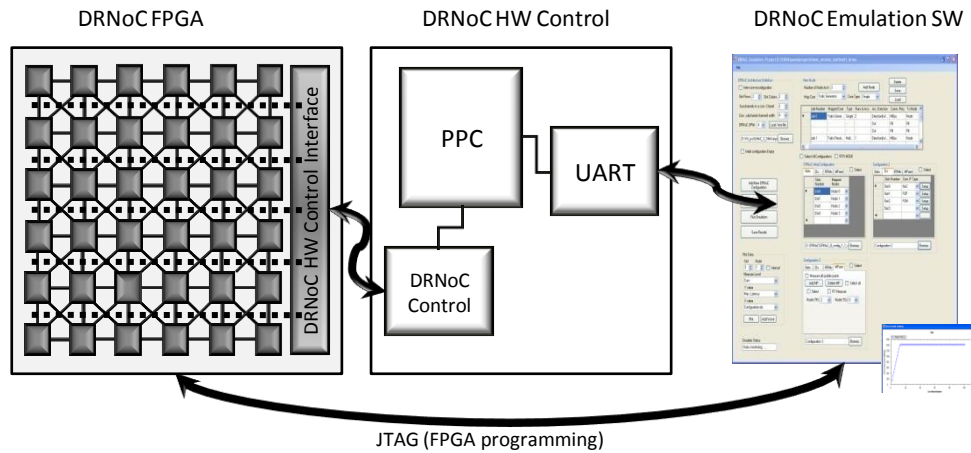


Figure 5.13: DRNoC emulation framework, divided in three platforms.

DRNoC node addressing: five for rows, five for columns, two for node slots selection (slot/cores, NI or RRM) and finally, two for internal registers selection. As a result, the same DRNoC control system (HW and SW) can be used with up to a 32x32 DRNoC and it can be easily adapted to other internal communication standards (no AMBA) by simply changing the communication bridge. Data is pulled out from each register consecutively and buffered in a FIFO allocated in the custom peripheral. If online measurement has been selected, data is constantly pulled out from the registers and buffered in the FIFOs from where they are constantly read by the PPC and sent to the Host PC. When the emulation process ends, the DRNoC FPGA generates an interrupt. The DRNoC HW control makes a final read of all registers and, after that, it interrupts the PPC. Differently, if the interest is put on max/min values, then appropriate TRs and NIs have to be included on the DRNoC. In this case, data is pulled out from the measurement registers only when the DRNoC FPGA interrupt has been received. After all data is pulled, the interrupt is retransmitted to the PPC that triggers the host PC DRNoC SW to read all data from the DRNoC HW control.

For having a time reference in the measurement system, a timer has been included in each TG, NI and TR. All timers are synchronized with the run command. The limit of the emulation system time is defined by these timers. In the current implementation, these timers are 32 bits, equivalent to 4G clock cycles. The DRNoC FPGA runs at 25 MHz resulting in 172 seconds total emulation time.

Regarding the data pulled from the measurement registers, the limit factor is the XUP board (HW control) to Host PC connection that is currently implemented by a serial link (RS232). The parallel DRNoC FPGA to XUP board link has a throughput of 65 Mbps and the PPC FIFO depth is 16K words, but the Host PC to XUP serial link is 9600 bps. Therefore, the system can pull and save in the host PC up to 10000 measured values

before it runs out of FIFO space. This problem can be easily solved by switching from the serial link to the Ethernet connection available in the XUP board.

5.5.3 DRNoC Emulation SW

A DRNoC Emulation control tool has been designed and runs on the measurement system host PC. The tool is in charge of controlling the entire emulation process following the work flow presented in section 5.5.1 and to administrate measured data and configurations. A screen shot of the software can be seen in Figure 5.14 and the tool main features are listed next:

1. The SW main function is to organize all configurations and data involved on the design space exploration process.
2. To define the application CTG and the DRNoC VA.
3. To define DRNoC configurations. The CTG mapping and the communication scheme definition.
4. To control the measurement process with the commands: run, stop, reconfigure and continue. The control is performed through the DRNoC HW control.
5. To prepare FPGA partial bitstreams. For this aim, the BITPOS and pBITPOS tools, described in Chapter 4, have been integrated in the DRNoC emulation control SW. The description files, needed by the tools, are automatically generated from the DRNoC architecture definition. BITPOS is used when the system full FPGA configuration includes hard cores. In this case, BITPOS is executed to extract hard cores and small bit manipulation related files, used for Intra-Core reconfiguration. The files are added to the local hard core library. Differently, pBTIPOS is used to properly allocate and reallocate hard cores during the DRNoC configuration load process.
6. The tool prepares all the needed scripts to configure both FPGAs and calls the FPGA programming tool (iMPACT) to program the FPGA through the JTAG port. Differently from the application presented in section 5.3, in this case the FPGA ICAP configuration port has not been used, because additional FPGA resources would be required for that and also, because the main point here is not system self-reconfigurability, but validation and demonstration of the emulation framework process.
7. To prepare raw data, pulled out from the DRNoC system, to be plotted (organize pulled data, calculate throughput and injection rate).

The tool works with hard cores that are held in the library. To generate hard cores for the library, the proposed hardware core design flow has been followed. For instance, if a new NoC router is going to be added, first a DRNoC with the proper options has to be generated with the DRNoCGEN tool (a tool, presented in section 3.4.4, that generates DRNoC compatible NoC models in VHDL).

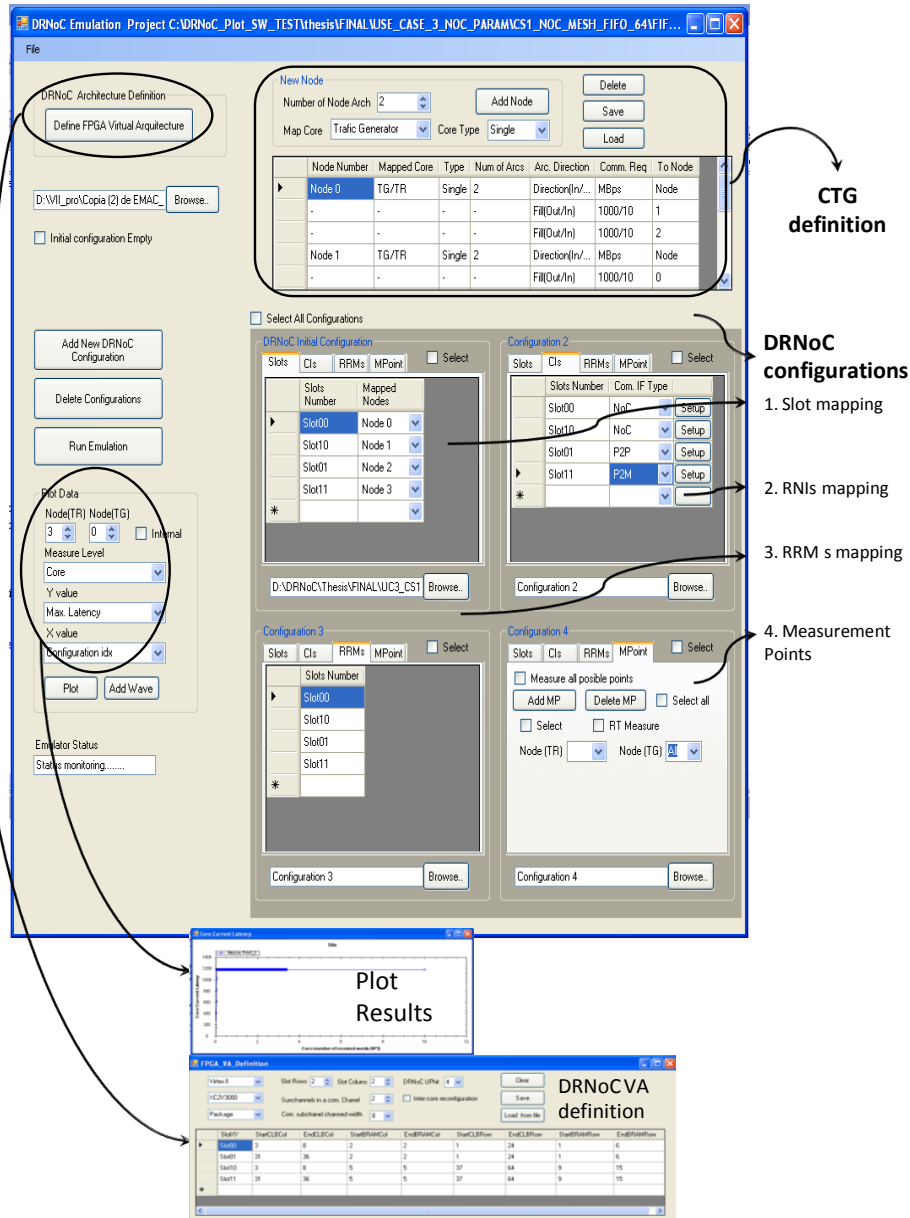


Figure 5.14: DRNoC Emulation SW. The SW is in charge of controlling DRNoC configurations and generating appropriate FPGA bitstream files. The tool is based on the work flow for design space exploration using partial reconfiguration, proposed in Chapter 4 and The BITPOS and pBITPOS tools.

The generated model includes more than a router, apart from the TGs, TRs and NIs. For having fast synthesis and since only one router is needed, an appropriate router is selected and isolated. As the code generated by DRNoCGEN follows the templates proposed in Chapter 2, this step is quite easy. Only the top and the user constraints file have to be modified (all but the router and the communication macros have to be commented). Then, a partial configuration file containing only the router hard core is extracted with BITPOS and added in the library along with a related description file. In both cases, synthesis times are reduced compared to entire system synthesise.

5.5.4 DRNoC Use Cases and Results

The objective of the use cases presented in this subsection is not to come up with the conclusion of which is the best communication scheme for each modeled applications CTG. Differently, the use cases have been selected in order to test and demonstrate the proposed workflow, the DRNoC architecture reconfigurability and to evaluate the feasibility of the proposed partial reconfiguration based emulation approach.

Next, a brief description of each use case can be found and, afterwards, all parameter values that will be used along the section are highlighted.

This subsection presents three use cases related to the emulation framework and DRNoC reconfigurability tests. A picture of the emulation framework setup can be seen in Figure 5.15. On the left side, the XUP board used for the DRNoC HW control and, on the right side, the DRNoC Architecture mapped on the FPGA of the custom development board that is also used in the section 5.3 application and has been specially designed for partial runtime reconfigurable systems implementation.

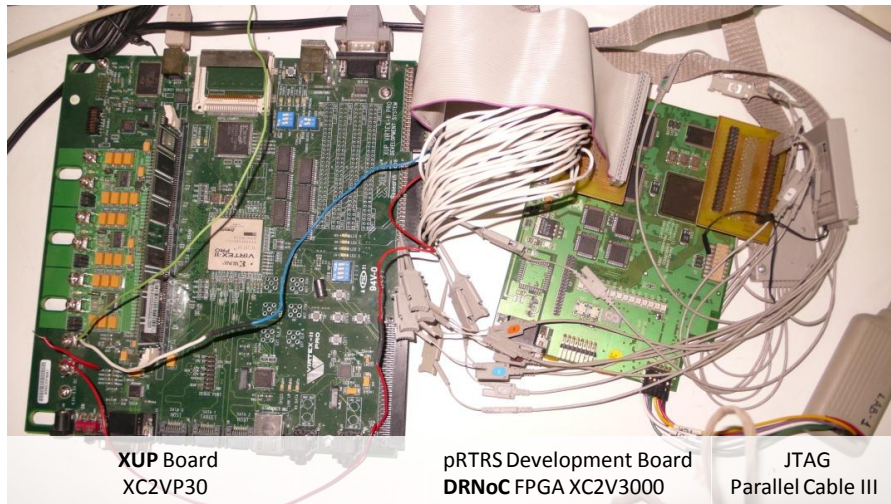


Figure 5.15: Picture of the Emulation Framework boards. XUP board on the left, a DRNoC mapped on the proprietary development board on the right. The JTAG parallel cable III can be also seen in the picture.

The main goal of the first use case is to test the context switching between different communication schemes, in an application design space, by using Inter-Core reconfiguration. RNI, RRM and RRM SMs are involved in this reconfiguration process and corresponds to changes in all the four NoC protocol layer implemented in HW. The selected application for the first use case (UC1) is a pipeline like application with two configuration that have been defined, emulated and briefly analyzed.

The second use case (UC2) is oriented to modify a NoC communication scheme, more specifically to change its topology. This process involves Inter-Core reconfiguration, where RRM are reconfigured, needed to modify the NoC protocol stack physical layer. Additionally, in this use case, Intra-Core reconfiguration has been used to change the measurement point tracking node in the traffic receiver node. Again, two configurations have been defined, emulated and briefly analyzed and compared.

The last use case (UC3) is oriented to test Intra-Core reconfiguration for tuning a specific NoC communication scheme parameter. In this case, the buffer size is changed and again, Intra-Core reconfiguration has been used to switch between different measurement point tracking nodes.

It is important at this point to remark that there is not a standardized benchmark for NoCs, although a lot of efforts have been put by the research community in that direction. Therefore, DRNoC communication schemes used along the use cases will not be compared with other conventional NoCs. Anyway, the performance parameters, normally used for NoC characterization, described in section 3.5.1, will be used to evaluate different DRNoC communication schemes. Furthermore, cost parameters are applied to evaluate and compare different implementation options (configurations).

Following the defined work flow, first an initial full configuration that contains the virtual architecture definition and, if it is the case, some cores allocated in each slot, is loaded in the FPGA. If the initial configuration does not include cores, partial reconfiguration is used to build the first configuration communication scheme by loading the appropriate hard cores. After that, partial reconfiguration has been used to pass to every subsequent configuration by modifying only the required slots or part of them.

All data needed for making the use case explanations more clear and performing the required calculations is presented in the next subsection, where an overall evaluation of the DRNoC design resources from the reconfigurability point of view has been included.

5.5.4.1 DRNoCs Design Resources Cost Evaluation

Cost parameters are directly linked to a DRNoC architecture implementation. As it has been mentioned several times, a 2x2 DRNoC for an XC2V3000 FPGA with the characteristics, mentioned in section 3.4.2 and Table 5.1 and summarized next is being used:

- The real slot/core size is 125 CLBs (500 slices).
- The real RNI size is 105 CLBs (420 slices).
- The real RRM size is 238 CLBs (952 slices) and each RRM have access to 6 BRAMs.

- Each channels width is 40 wires in each direction and is divided into 4 subchannels (a total of 80 wires are available in each channel).
- Each RRM has 3 channels that are connected to all the neighboring RRMs in the 2x2 DRNoC.
- Each local channel consists of two channels in each direction of 40 wires each. A total of 160 wires are available in the local channel.

Based on the previously listed values, implementation cost parameters for hard cores to be used along the use cases are presented in Table 5.9 and for making it more clear, a description of each hard core can be found in Table 5.8.

Table 5.8: Use cases hard cores description.

Cores	description	Reconfig.
TG-S	A traffic generator core that generates traffic to a single receive (Simple TG)	Yes
TG-M	A traffic generator core that generates traffic to multiple receives	Yes
TR-MP	A traffic receiver that includes a single measurement point	No
R-XY-8b2C	Router for an 8bit flit XY NoC that uses 2 channels	Yes
R-XY-8b3C	Router for an 8bit flit XY NoC that uses 3 channels	Yes
R-XY-16b2C	Router for a 16bit flit XY NoC that uses 2 channels	Yes
R-FT-8b1C	Router feed-through for an 8bit flit NoC that uses 1 channel	Yes
R-P2P-UD-FT-73w-1C	RRM P2P unidirectional feed-through that uses 1 channel and 73 wires	Yes
R-P2P-BD-FT-73w-2C	RRM P2P bidirectional feed-through that uses 2 channels and 73 wires from each one	Yes
NI-UD-FT-73w	RNI AMBA Master-Slave feed-through link	Yes
NI-BD-FT-146w	RNI AMBA full duplex feed-through link	Yes
NI-TG-XY-8b	TG Network Interface for an 8 bit flit XY NoC	No
NI-TG-XY-16b	TG Network Interface for an 16 bit flit XY NoC	No
NI-TRMP-XY-8b	TR Network Interface for an 8 bit flit XY NoC with a single online measurement point	No
NI-TRMP-XY-16b	TR Network Interface for an 16 bit flit XY NoC with a single online measurement point	No

Hard cores naming is directly related to the number of used channels and/or channel wires (see Table 5.8). Notice that the local channel is not accounted neither in the name acronyms nor in the channels cost.

As it has been mentioned several times, the restrictions of current place and route tools (defining routing restrictions is not possible) makes very difficult to keep all routing wires inside a defined bounding box. Furthermore, if at least a single wire crosses slot boundaries, the resulting core cannot be relocated. A lot of efforts have been put to restrict the routing and permit all cores to be relocatable, but it has not been possible for cores that occupy more than 55 % of the FPGA, like any TRs with measurement point(s)

and also for NIs. The consequence of this is that such cores cannot be included in the general hard core library. In Table 5.8, a column that specifies if the core is relocatable and has been included in the library can be found (Reconfig. column).

To overcome this problem along the further discussed use cases, slot borders have been moved in some cases and, in others, cores have been partially and locally grouped. For instance, a core/slot to RNI boundary could be blurred by moving the border with few columns (without modifying the bus macro positions) in order to have more routing room. Some of these cores are considered as entire units, for instance a core with a fixed NI, while others are considered as smaller hard cores, cores that occupy less than a slot, a small NI for instance. If such situation appears, all affected cores are included only in a local library, specific for an emulation project.

Concerning the use cases presented afterwards, in the first two use cases (UC1 and UC2) such cores are loaded with the initial configuration and afterwards, partial configuration is used to load hard cores from the general library, contrary, the last use case (UC3) tests Intra-Core reconfiguration and the mentioned problem does not appears.

Table 5.9: Use cases hard cores cost parameters.

Cores	Cw_{ij}	Cc_i	$Area$ <i>slices</i>	Ca_i	Men	Cm_i
R-XY-8b2C	0.25	0.66	262	0.275	3	0.5
R-XY-8b3C	0.25	1	300	0.315	4	0.666
R-XY-16b2C	0.45	0.66	360	0.378	3	0.5
R-FT-8b1C	0.25	0.22	0	0	0	0
R-P2P-UD-FT-73w-1C	0.912	0.22	0	0	0	0
R-P2P-BD-FT-73w-2C	0.912	0.66	0	0	0	0
NI-UD-FT-73w	0.456	1	0	0	0	0
NI-BD-FT-146w	0.912	1	0	0	0	0
NI-TG-XY-8b	0.125	1	230	0.547	0	0
NI-TG-XY-16b	0.225	1	241	0.573	0	0
NI-TRMP-XY-8b	0.125	1	232	0.552	0	0
NI-TRMP-XY-16b	0.225	1	244	0.58	0	0

In the next paragraphs, cost parameters values that will be required during the use cases communication options analysis are described.

Implementation cost parameters have been calculated and included in Table 5.9 for all hard cores prepared to be loaded in RRM and RNI. Although most of the defined cost parameters are applicable to cores to be loaded in slots (TG and TRs), they are not included in the Table, as they are not part of the communication structures building elements.

Values included in Table 5.9 are related to : i) Cw_{ij} that is the communication wires

cost, ii) Cc_i is the communication channels cost, iii) $Area$ is the node element needed area, iv) Ca_i is the node element area cost, calculated with respect to the assigned slot real area, v) Men is the node element used memory and vi) Cm_i is the core memory cost.

It can be noticed from the Table that the most costly cores in terms of wires are all the related to P2P communications, like R-P2P-BD-FT-73w-2C and NI-BD-FT, but they have the lowest area cost. On the other side, the highest area cost and lowest wire cost cores are for the cores associated to NoC communication schemes, like R-XY-8b3C and NI-TRMP-XY-8b.

Based on the cost values from Table 5.9, it is possible to calculate communication schemes total implementation cost and select the less costly communication strategy that covers the application performance requirements, build it and load it in the reconfigurable system. Total cost values have been calculated for each configuration used in the next subsection. Furthermore, cost parameters can be used to know how many communication schemes can coexist in a DRNoC architecture. Theoretically, any amount of communication schemes can coexist, as long as the resulting total cost parameters for each element is <1 .

Reconfiguration costs for each slot in the current DRNoC implementation can be found in Table 5.10. The Table includes configuration file size for each slot, and also the needed time to download each file through the JTAG interface with the Xilinx programming tool (iMPACT) using a Parallel cable IV. The Table also includes data related to Intra-Core reconfiguration of LUTs and BRAM content data that is independent from the slot size. Cost values calculation is related to the time needed to reconfigure the entire FPGA (that is 4 seconds) and the full configuration file size is 1.28 MB. Configuration time values included in the Table have been measured with an oscilloscope measuring the JTAG clock (TCK) activity period. It can be noticed from the reported data that reconfiguration time differences are not proportional to file size differences. For instance, 0.5 seconds were measured for 3K file download, while 2 sec. were measured for the 91 KB files size. This could be due to the SW delay and the needed JTAG on-chip controller setup time.

Values from this Table will be further used to calculate the total reconfiguration cost for each configuration.

Table 5.10: Slots reconfiguration cost and time for Intra and Inter-Core reconfigurations.

Slot	file size KB	T_{reconf} sec.	C_{reconf} -
Full FPGA	1280	4	-
slot/core	91	1.5	0.378
RNI	91	1.5	0.378
RRM	166	2	0.4
LUTs	3	0.5	0.125
BRAM	50	1.2	0.300

5.5.4.2 Use Case 1: Inter-Core Reconfiguration to Change the Communication Scheme

This first use case (UC1) is a piece of a pipeline based application where a task generates data irregularly that is saved in an intermediate FIFO from where it is pulled out immediately (contentions are not foreseen). Thus, the application is made up by three tasks, see Figure 5.16 (upper part).

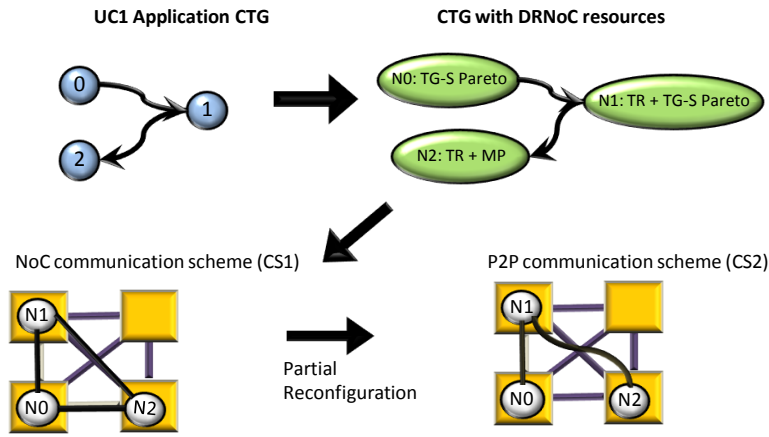


Figure 5.16: Test of Inter-Core reconfiguration used to change the communication scheme. The application CTG is shown on the top left edge of the Figure. Its DRNoC based model is shown on the top right corner and two configurations are defined and shown in the bottom part: one is a pure NoC (CS1) on the left and the other one is a pure point to point (P2P) on the right (CS2).

Each task has been modeled with a DRNoC design resource as follows:

- The task that generates traffic to the FIFO (node0) has been modeled with a single Pareto traffic generator (N0:TG-S-Pareto in Figure 5.16).
- The FIFO and the data pulling has been modeled with a TR and a TG (TG-TR) (node1), the TG generates traffic with the same distribution and throughput as node0, but with some initial delay (N1: TR-TG-S-Pareto in Figure 5.16)
- The pulled data is collected in Node2 that is modeled as a traffic receives and includes a measurement point (N2: TR-MP in Figure 5.16).

Each traffic generator has been configured to send 10000 packets of 10 words each (1 word is 32 bits) simulating an MPEG 2 burst distribution. Afterwards, two more words are added in the NI. One corresponds to the packet header that includes a last packet flag and the second one is a time stamp used by the measurement system. A single measurement point has been included in node2 since the traffic performance of both links (N0-N1 and N1-N2) is the same.

Once the CTG has been modeled with DRNoC design resources (the first step of the work flow), configurations are assigned. Two configurations with the same mapping to

the DRNoC architecture (node0 to slot00, node1 to slot01 and node2 to slot10), but with different communication schemes have been defined. The First communication scheme (CS1 in the bottom left side of Figure 5.16), included in the FPGA initial configuration, is an 8 bit flit and 4 bit uphit NoC. The NoC is build by (see Table 5.8 for acronyms):

- Two NI-TG-XY-8b loaded in RNI00 and RNI01.
- One NI-TRMP-XY-8b in RNI10.
- Three R-XY-8b2C loaded in RRM00, RRM01 and RRM10.

The second communication scheme (CS2 in the bottom right side of Figure 5.16) is a pure P2P built by:

- One NI-BD-FT-146w loaded in RNI01.
- Two NI-UD-FT-73w loaded in RNI00 and in RNI10.
- Two R-P2P-UD-FT-73w-1C loaded in RRM00 and RRM10.
- One R-P2P-BD-FT-73w-2C loaded in RRM01.

Notice that, in the NoC configuration, the RRM01 and RRM10 switch matrixes have been modified to redirect respectively the south and north router port and take advantage of the Xmesh diagonal. Otherwise, an additional router would be needed in RRM11 and this would increase NoC latency and total area overhead.

As example, the FPGA Editor DRNoC 2D floorplanning of each configuration, CS1 on the left side and CS2 on the right, are presented in Figure 5.17. Based on these floorplannings, hard cores for the general and the local library have been generated. From the CS1 floorplanning, hard cores that correspond to the NoC routers are extracted for the general library, while from CS2, RRM feed-throughs.

For switching from CS1 to CS2 and vice versa, following the proposed work flow in section 5.5.1 - Figure 5.12, four partial reconfigurations are required: two to modify RNIs and a two more to modify RRM. Thus, using data from Table 5.10, the total reconfiguration cost for any configuration is equal to $C_{totalreconf} = 1.55$. This value could be further used in a control system to decide if a reconfiguration is worthy.

Total cost parameters for building both configuration schemes have been calculated following the equation presented in 3.6.3 with data from Table 5.9 and can be found in Table 5.11. The NoC configuration (CS1) total cost is more than double, compared to the P2P one (CS2), but wire requirements (C_{totalw}) of CS1 are four times less. As reference it can be taken into account that the maximum value of the total wires cost for any communication scheme that uses 3 nodes is 6 (the same value as the elements required for building it). Anyway, if the wires cost values are the leading parameter for selecting the communication scheme, then CS1 is a better choice. On the contrary, if area is more important, CS2 is the best choice.

After both configurations have been defined and emulated, latency and throughput results have been plotted. A brief analysis can be found in the next paragraphs and the plotting screen shots can be found at the end of this use case description.

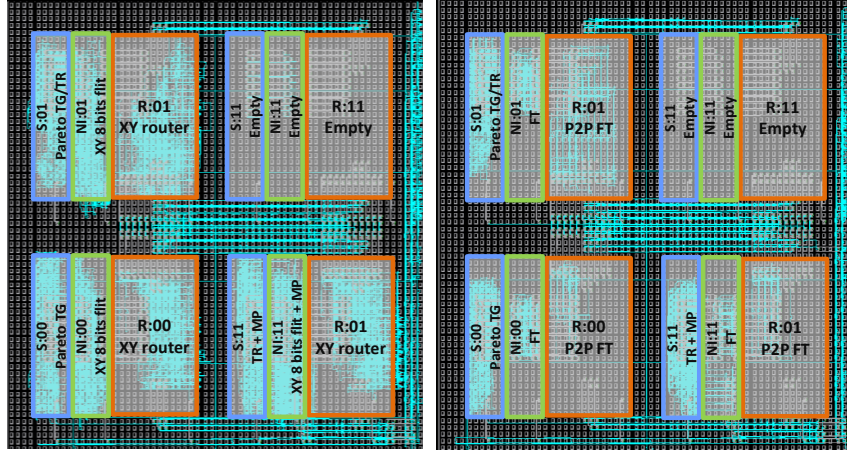


Figure 5.17: FPGA Editor floorplanning for CS1 (NoC) on the left and CS2 (P2P) on the right.

Table 5.11: Total cost parameters for two communication schemes: one P2P (CS1) and one NoC (CS2) for the pipeline application (Use Case 1-UC1).

CS	C_{totalw}	C_{totalc}	C_{totala}	C_{totalm}	C_{totali}
CS1	1.25	4.98	2.953	1.5	10.683
CS2	4.548	0	0	0	4.548

For CS1 (NoC), latency and throughput online measurements in the TR core (node2) and the same node NI-TR can be found in Figure 5.18 and Figure 5.19 respectively for the first 500 packet transmission. The first aspect to be noticed is data transmissions burst behavior. During a transmission period, latency constantly increases, but the NoC contention threshold is not reached and therefore current throughput remains constant during burst ON periods. The throughput low peaks that appear in the plots are because, during the burst OFF periods, when data is not being received, the time between two consecutive packets reception is very high.

For CS2 (P2P), the online latency and throughput, measured in the same point, and also for the first 500 packets can be seen in Figure 5.20 (all Figures are included at the end of the section). Latency is constant and burst independent as the communication scheme uses the AMBA bus protocol that has 2 cycle latency. In this case, latency is measured from the appearance of the bus select signal until the end of a word transaction, after the slave ready response signal. On the contrary, the TG core burst behavior can be noticed in the throughput plot and the reasons for the low peaks, like in CS1, are again the burst OFF periods.

Maximum and minimum latency and throughput values can be found for both configurations in Table 5.12. The CS1 min. and max. latency, measured at core level, is higher than the NI latency (almost 30 cycles). This is because 6 cycles are spend in both, the TG core to NI-TG interface and the data serialization process at the source node,

and 24 cycles are needed in the NI-TR receiver. The NI-TR first has to discard the packet header and the time stamp, put by the NI-TG and then it has to receive and prepare a complete word previously to transmit it to the TR. The same reasoning is applicable for the remaining NoC use cases, therefore it will not be explained again.

Regarding the CS2, the latency is constant and the maximum throughput value is 0.3 words/cycle because there is one waiting clock cycle between two consecutive AMBA transmissions (that have 2 cycle latency).

Table 5.12: Maximum and minimum latency and throughput values for two communication structure CS1(NoC) and CS2 (P2P) configuration for UC1.

CS	Core Latency cycles		Core Throughput words/cycle		NI Latency cycles		NI Throughput words/cycle	
	max.	min.	max.	min.	max.	min.	max.	min.
CS1	286	47	0.09	0.0077	190	19	0.4	0.036
CS2	2	2	0.3	-	-	-	-	-

The implementation cost results, along with the performance results, have shown that the best option for this use case is CS2, the P2P communication scheme. Of course if there are enough free wires to load it. Anyway, further configurations, like a P2P with lower wires cost can be added to the emulation system project and be emulated.

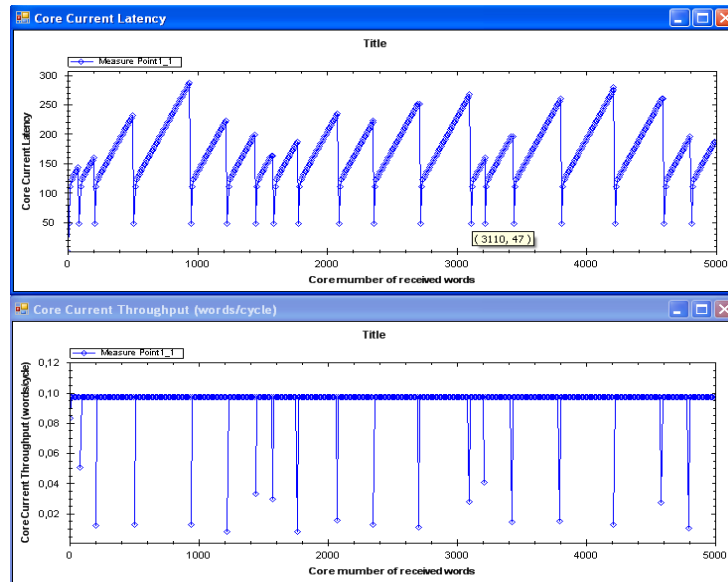


Figure 5.18: CS1 (NoC) current latency and current throughput, measured at core level. The burst behavior can be clearly seen in both curves.

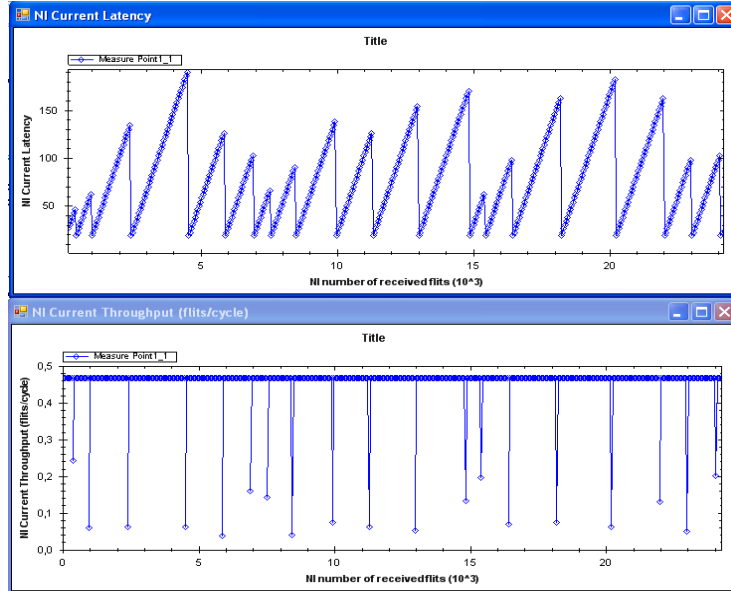


Figure 5.19: CS1 (NoC) current latency and current throughput, measured at NI level. The burst behavior can be seen in both curves.

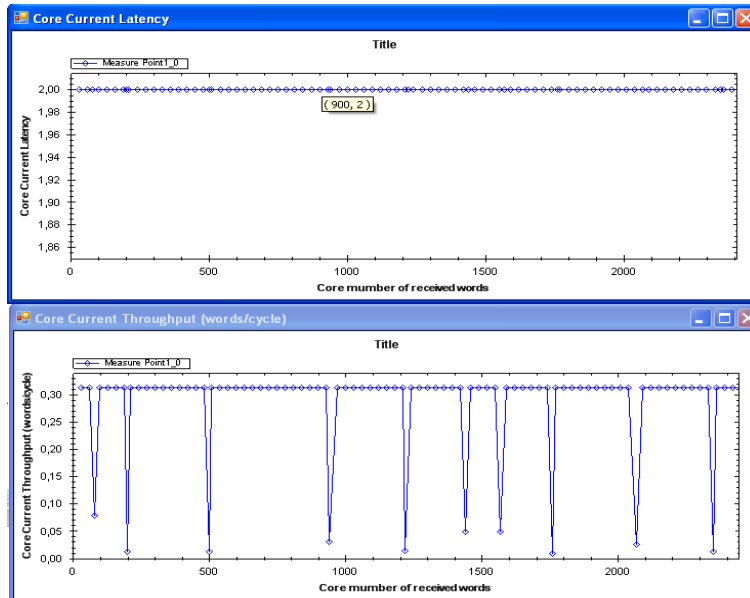


Figure 5.20: CS2 (P2P) current latency and current throughput, measured at core level. The burst behavior can be seen in the throughput, while the latency is constant.

5.5.4.3 Use Case 2: Inter-Core Reconfiguration of NoC Physical Layer

This use case main goal is to test Inter-Core reconfiguration for modifying NoCs physical layer. The small application selected and modeled, shown on Figure 5.21, is a very common situation where several cores, in this case three, try to access the same media.

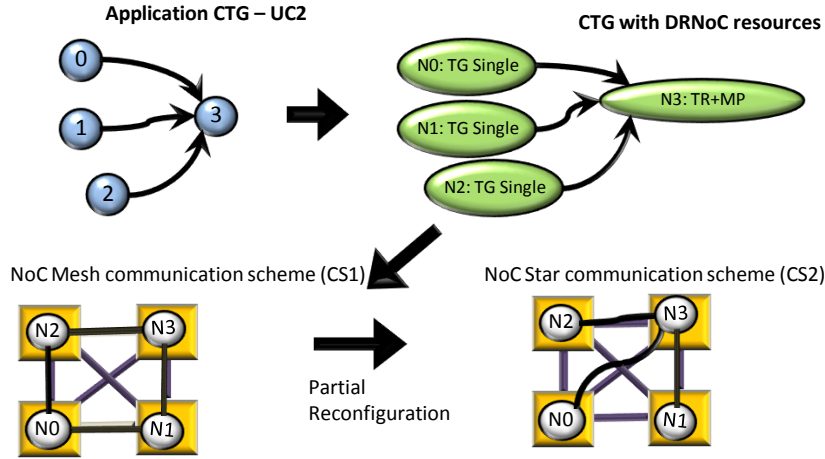


Figure 5.21: Test of Inter-Core reconfiguration to change a communication scheme physical layer. The application CTG is shown on the top part of the Figure, along with its DRNoC mapping. Two communication schemes configurations for this application can be seen on the bottom part of the Figure: one Mesh NoC (CS1 on the left side) and one Star NoC (CS2 on the right side).

The application, composed of four nodes (see upper left corner of Figure 5.21) has been modeled as follows:

- The common media is a traffic receiver (node3) with a single measurement point (N3: TR-MP in Figure 5.21).
- The remaining cores are modeled as regular traffic generators (TG Single), node0, node1 and node2.

Each traffic generator has been configured to send 10000 packets of 10 words each (1 word is 32 bits) with the highest possible injection rate (no wait clock cycles between packets). Furthermore, a single measurement point has been included in node3 that afterwards, using Intra-Core configurations, has been modified to track different TG nodes.

Two configurations, with the same DRNoC architecture mapping (node0 to slot00, node1 to slot10, node2 to slot01 and node3 to slot11) and with different communication schemes have been defined for this use case. The First communication scheme (CS1), included in the FPGA initial configuration, is an 8 bit flit and 4 bit uphit mesh topology NoC. The NoC is built by (see Table 5.8 for acronyms):

- Three NI-TG-XY-8b loaded in RNI00, RNI10 and RNI01.
- One NI-TRMP-XY-8b loaded in RNI11.
- Four R-XY-8b2C loaded in every RRM.

The second configuration scheme (CS2) communication is an 8 bit flit and 4 bit uphit star topology NoC that makes use of an Xmesh diagonal. It is built by:

- Three NI-TG-XY-8b loaded in RNI00, RNI10 and RNI01.
- One NI-TRMP-XY-8b loaded in RNI11.
- Three R-FT-8b1C loaded in RRM00, RRM01 and RRM10 and finally one R-XY-8b3C loaded in RRM11.

For switching from CS1 to CS2 and vice versa, two Inter-Core partial reconfigurations of RRM are required. As a result, the total reconfiguration cost for passing from CS1 to CS2 and vice versa is $C_{totalreconf} = 0.8$ (less than in UC1 that was 1.55). Further, Intra-Core partial reconfiguration has been used to change the measurement point tracked TG.

Total cost parameters, calculated following the equation presented in section 3.6.3 with data from Table 5.9, for constructing both configuration schemes can be found in Table 5.13. The mesh NoC configuration (CS1) total implementation costs (C_{totali}), as well as all partial costs, are around 25 % higher compared to the star configuration (CS2). Therefore, at this stage, it could be said that for this use case the best option is CS2. Nevertheless, NoCs performance has also to be taken into account and therefore it is briefly analyzed in the next paragraphs.

Table 5.13: Total cost parameters for the mesh NoC (CS1) and the star NoC (CS2) of UC2.

CS	C_{totalw}	C_{totale}	C_{totala}	C_{totalm}	C_{totali}
CS1	1.5	6.64	3.018	2	12.344
CS2	1.125	5.66	2.508	0.666	9.959

After both configurations have been defined and emulated, current latency and current throughput online measures plots can be found for CS1 (mesh NoC) in Figure 5.22 for core level measurement and in Figure 5.23 for NI measurement. The defined measurement point, shown in all the figures and allocated in node3, tracks data received from node0 (MP3-0) that is the longest path in the mesh NoC. From this figures it can be noticed that latency linearly increases (because router buffers are being filled) until reaching a constant value that is the NoC latency for this constant traffic pattern. Notice that figures 5.22 and 5.23 show the latency for the first 160 packets that corresponds to its rising edge. At the end of the transmission of node1 to node2, directly connected to node3, latency decreases again. Differently, the current throughput is constant. The time interval between two consecutive packets is the same, because the traffic distribution in the NoC is regular.

For the star NoC example, online current latency and current throughput, measured at core and NI level, respectively, can be seen in Figure 5.24 and Figure 5.25 for the same measurement point (MP3-0) and also for the first 160 packets transmission. Latency, like in CS1, rapidly increases until reaching the maximum value and afterwards, it decreases when the NoC load is reduced (because the transmission of data from node0 to node 2 finishes). The throughput, shown in the lower part of the same figures, is constant, like in CS1.

To summarize, maximum and minimum latency and throughput values, measured at core and NI level, can be found for both configurations in Table 5.14. The table does not include throughput min. values as they are constant and have been included in the table as max. values. Data in the Table makes clear that CS2 has lower latency and higher throughput, compared to CS1, and this confirms the statement that CS2 is a better choice for this use case.

Table 5.14: Max. and min. latency and throughput for CS1(mesh NoC) and CS2 (star NoC).

CS	Core Latency cycles		Core Throughput words/cycle		NI Latency cycles		NI Throughput words/cycle	
	max.	min.	max.	min.	max.	min.	max.	min.
CS1	3236	350	0.0247	-	2920	322	0.118	-
CS2	1179	141	0.033	-	879	113	0.158	-

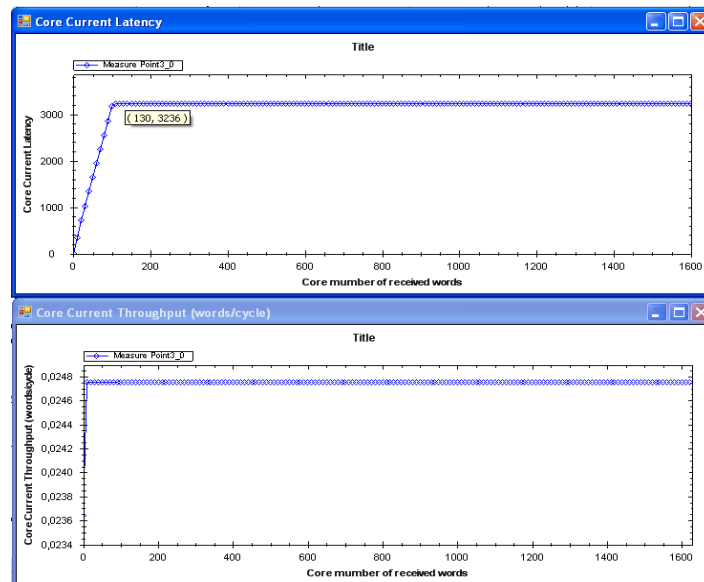


Figure 5.22: CS1 (mesh NoC) current latency and current throughput plot, measured at core level.

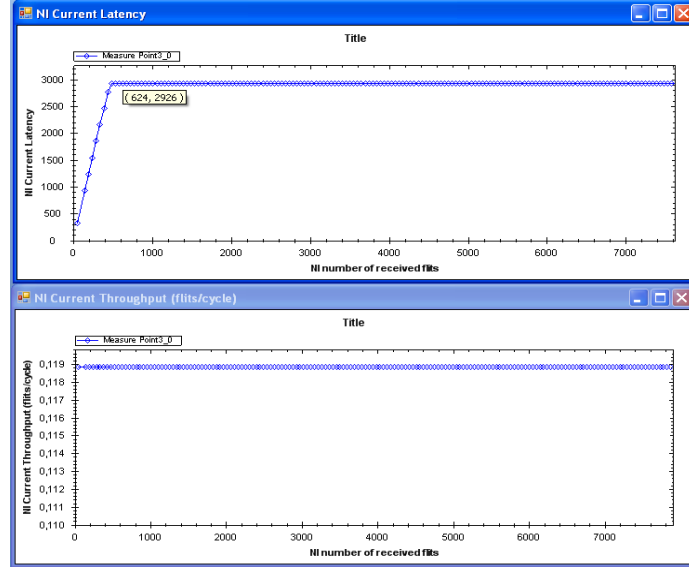


Figure 5.23: CS1 (mesh NoC) current latency and current throughput plot, measured at NI level.

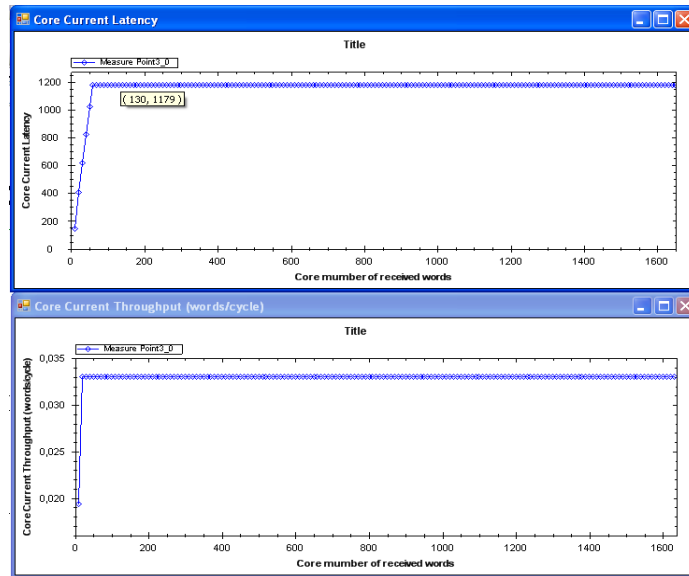


Figure 5.24: CS2 (star NoC) current latency and current throughput plot, measured at core level.

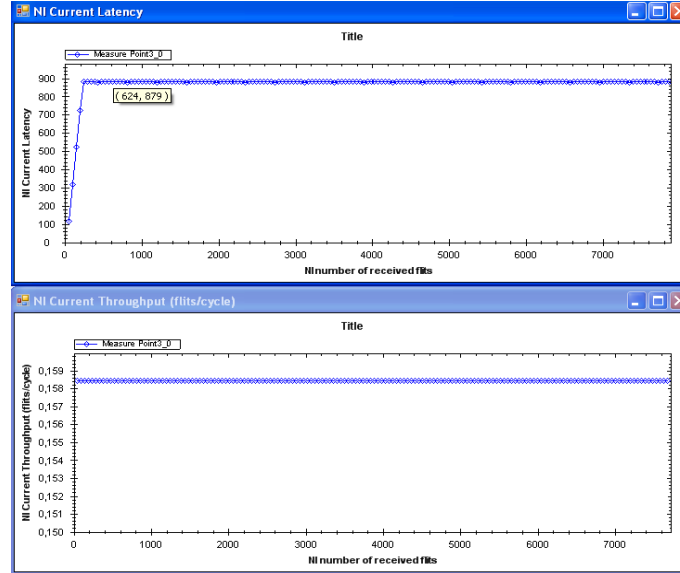


Figure 5.25: CS2 (star NoC) current latency and current throughput plot, measured at NI level.

5.5.4.4 Use Case 3: Intra-Core Reconfiguration to Change NoC Parameters

This use case is created to test Intra-Core reconfiguration for changing a communication scheme parameter. More specifically to change NoC routers FIFO buffers size: one with a FIFO of 64 entries (CS1) and another with a FIFO of 256 entries (CS2). Modifications are achieved by small bit Intra-Core reconfiguration that affects LUTs content. The communication task graph of the selected application is shown on the top left corner of Figure 5.26.

It is composed of four nodes that, differently from the previous use cases, do not follow a specific model for node communication. All nodes have been modeled as DRNoC regular traffic generators that send 10000 packets of 10 words, distributed in two groups of 5000 packets. The highest possible packet injection rate (no waiting cycles between packets) has been defined for the first group, while the second group has a lower rate (1000 waiting cycles between two packets).

Each TG addresses different, target, receiving nodes (see the top part of Figure 5.26):

- Node0 generates traffic to all the remaining nodes (node1, node2 and node3).
- node3 also generates traffic to all the remaining nodes (node0, node1 and node2).
- Node1 and node2 generate traffic to both, node0 and node3.

Additionally, a single measurement point (MP) has been included in node3.

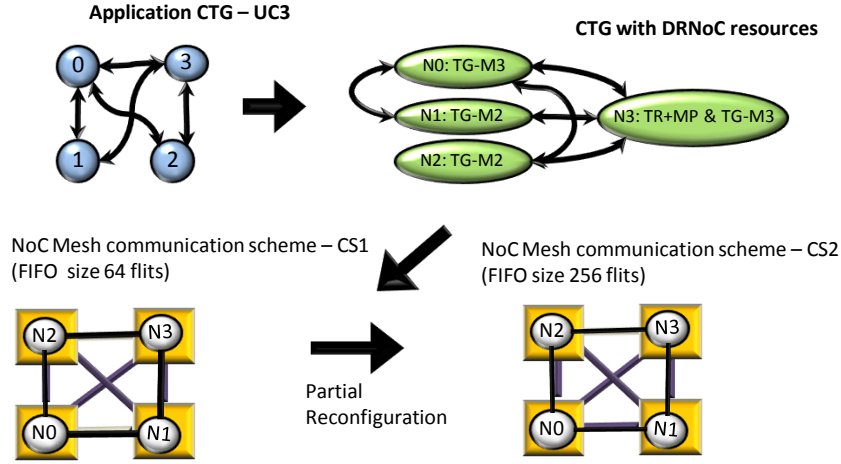


Figure 5.26: Test of Intra-Core reconfiguration used for changing a NoC parameter. An application CTG and the DRNoC model, made up by four different TG-TRs and one measurement point in node3, can be seen on the top part of the Figure. The application has two configurations, shown on the bottom part. Both have the same communication scheme (Mesh NoC), but different buffer size: one with a FIFO of 64 entries (CS1) and another with a FIFO of 256 entries (CS2).

Two configurations have been defined for the described application CTG. Both have the same mapping of nodes to slots: node0 to slot00, node1 to slot01, node2 to slot10 and node3 to slot11 and also both use the same NoC communication scheme, which is a 16 bit flit and 4 bit uphit mesh. The NoC is composed of (see Table 5.8 for acronyms):

- Four NI-TG-XY-16b allocated in RNI00, RNI01, RNI10 and RNI11.
- One NI-TRMP-XY-16b allocated in RNI11
- Four R-XY-16b2C allocated in RRM00, RRM01, RRM10 and RRM11.

The difference between both configurations is the FIFO buffer size. The first communication scheme (CS1) has a FIFO size of 64 flit entries, while the second communication scheme (CS2) has 256 flits entries FIFOs.

During the emulation process, Intra-Core partial reconfiguration has been used to pass from one configuration to the other and also, to change the measurement point tracking TG. Since three TGs generate traffic to node3, three partial reconfigurations are required and the HW emulation has to be run three times for each configuration emulation.

The total reconfiguration cost for passing from CS1 to CS2 and vice versa is $C_{totalreconf} = 0.25$ (two small bit LUTs configuration of 0.125 cost value each), the lowest one from all the use cases.

Total cost parameters for the defined configuration schemes, calculated using data from Table 5.9, can be found in Table 5.15. As both configurations use the same amount of resources, both have the same cost parameters (each buffer occupies one BRAM).

Table 5.15: Total cost parameters for the two mesh NoC communication strategies (CS1 and CS2) related to the application used in UC3.

CS	C_{totalw}	C_{totalc}	C_{totala}	C_{totalm}	C_{totali}
CS1	2.7	6.64	2.637	2	14.013
CS2	2.7	6.64	2.637	2	14.013

It can be remarked that the total communication scheme building cost is the highest, compared to the rest use cases as these NoCs have double the previously used flit width.

After both configurations have been emulated, current latency and current injection rate online measures can be found for CS1 (64 FIFO entries) in Figure 5.28 for measures at core level and in Figure 5.29 for the NI level. Similarly, plots for CS2 (256 FIFO entries) can be found in Figure 5.30 and Figure 5.31.

Plots shown in all Figures (that can be found at the end of the use case) correspond to the measurement point allocated in node3 (slot11) while tracking data received from node0, (MP3-0). Notice that, contrary to the previous use cases, here plots correspond to online injection rate and latency (not throughput and latency). Current injection rate is measured with respect to a delta time that is the time interval between two consecutive packets headers entering in the NoC.

Curve, show the latency and injection rate behavior during the transmission of the first 250 packets (part of first group of packets) from source node0 to target node3. During this time, the remaining nodes are also trying to send data at the highest possible rate to different receives. The TG to TR active links for the first 5000 packets are: node3 to node0, node2 to node3 and, node1 to node0.

An "uncommon situation" can be noticed from the current latency curves, shown in figures 5.28, 5.29, 5.30 and 5.31. The reached maximum latency by CS2 is higher compared to CS1 which has more than 4 times smaller buffers. The CS1 NoC needs more time to inject the same amount of packets, compared to the CS2 NoC, but packets in CS2 remain more time in the NoC waiting for being routed. Therefore the maximum latency value for CS2 is higher than for CS1. Furthermore, the CS1 NoC goes into contention much earlier than CS2. This can be noticed from figures 5.28, 5.29 and figures 5.30 and 5.31. The injection rate in CS2 (figures 5.30 and 5.31), remains in a maximum value for more time, compared to CS1 (figures 5.28 and 5.29). Also, there are more fluctuations in the CS1 injection rate compared to CS2. Actually CS2 and CS1 work at different operating points. Each configuration NoC latency to injection rate characteristic curve can be seen in Figure 5.27.

The characteristic curves have been measured under a controlled, testing, NoC traffic load. The approximate operating point for each NoC configuration has been marked in Figure 5.27. As it can be noticed, the CS2 NoC (256 entry FIFO) has a higher operating point, with almost 20 MBps higher total injection rate, compared to CS1 NoC (64 entry FIFO). This results in the CS2 NoC higher latency on the previously discussed graphics. On the contrary, if the NoCs were not working in the same working point, with the same injection rate, both would have the same, constant, latency. This can be perceived from Figure 5.27 and is because the longest NoC path has only one hop [MCM⁺04b].

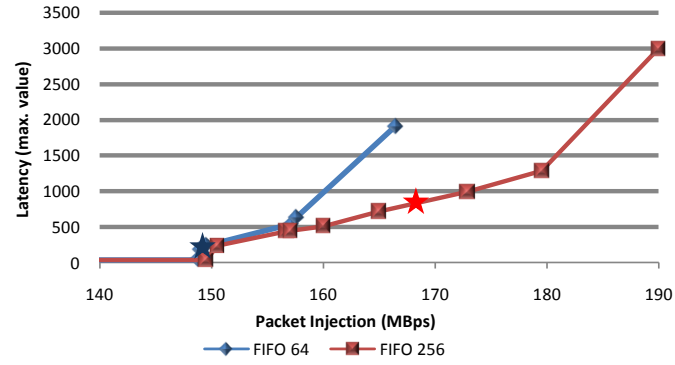


Figure 5.27: CS1 (64 entries FIFO) and CS2 (256 entries FIFO) NoC latency characterization.

A summary of the maximum and minimum current latency and current injection rate values, measured at core and NI level, can be found for both configurations in Table 5.16.

Table 5.16: Max./min. current latency and current injection rate values for two communication structures, CS1(64 entries FIFO) and CS2 (256 entries FIFO), for the UC3 application.

CS	Core Latency cycles		Core Injection Rate words/cycle		NI Latency cycles		NI Injection Rate words/cycle	
	max.	min.	max.	min.	max.	min.	max.	min.
CS1	198	31	0.196	0.178	169	26	0.470	0.421
CS2	625	31	0.196	0.176	606	26	0.470	0.413

Taking into account the presented results, it can be concluded that a better injection rate has been achieved with CS2. Selecting an appropriate buffer size is not a simple task. In a common situation, the NoC operation point changes during an application execution time and thus the buffers demand. Partial reconfiguration of the buffers' size in execution time could be a solution to have a good NoC area/performance tradeoff.

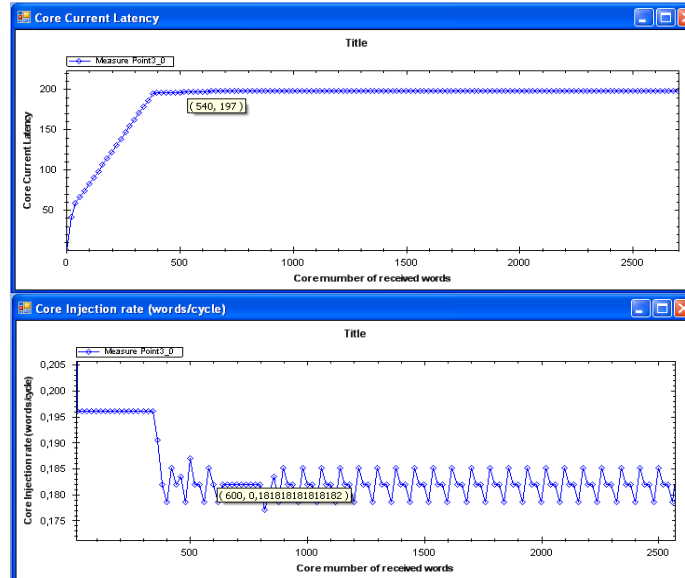


Figure 5.28: CS1 (64 entries FIFO) current Latency and injection rate plot, measured at core level.

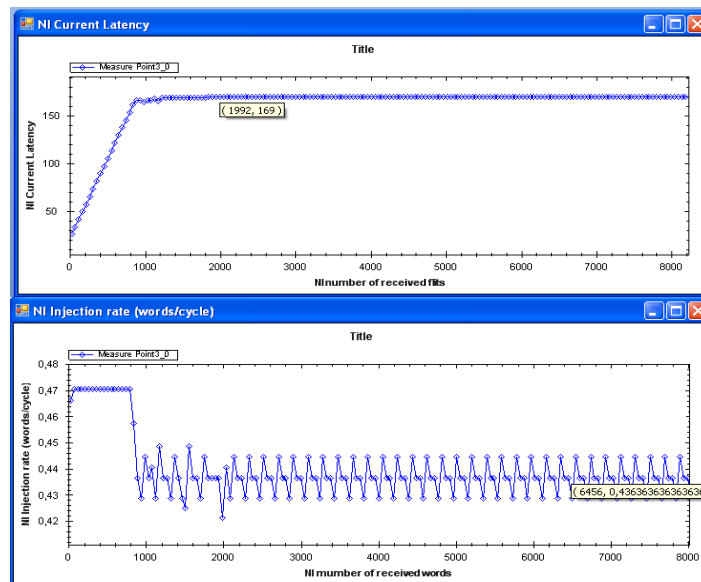


Figure 5.29: CS1 (64 entries FIFO) current Latency and injection rate plot, measured at NI level.

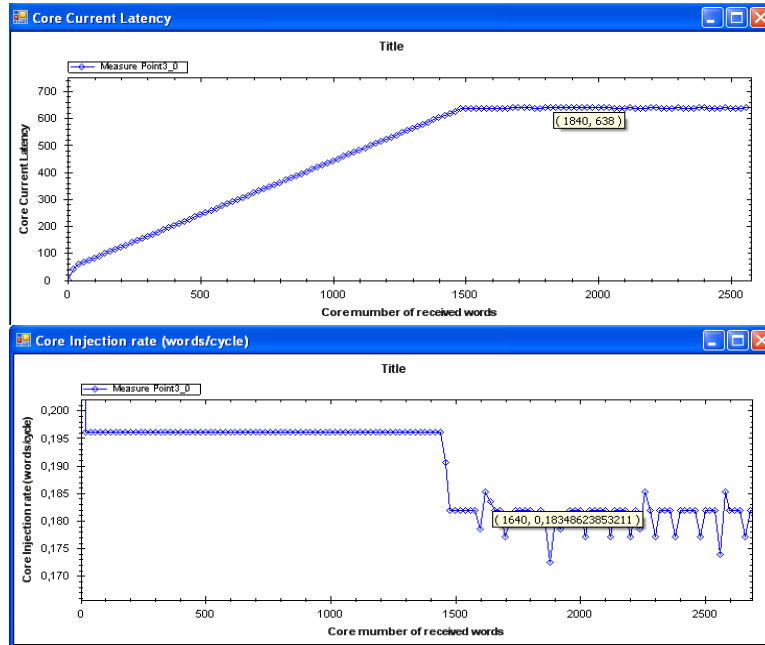


Figure 5.30: CS2 (256 entries FIFO) current Latency and injection rate plot, measured at core level.

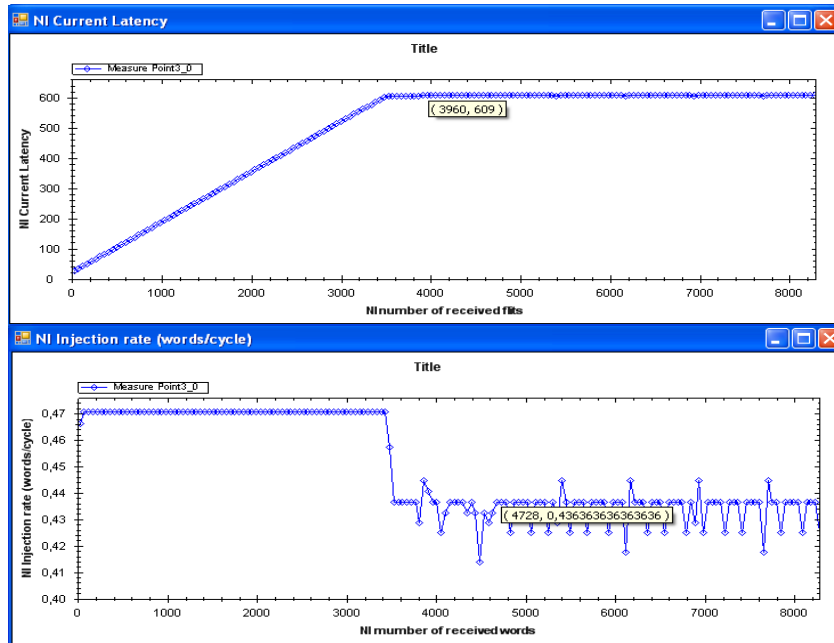


Figure 5.31: CS2 (256 entries FIFO) current Latency and injection rate plot, measured at NI level.

5.5.5 Evaluation of the Proposed Emulation System

This section evaluates the proposed solution by reporting the achieved speedups compared to a non partial reconfiguration solution and highlighting its advantages and disadvantages.

5.5.5.1 Speedup Analysis

To evaluate the proposed solution for NoC emulation, the use cases presented in this section have been compared with a flow that has the same steps but, instead of using partial reconfiguration, it is fully based on conventional synthesis. Speedup results have been included in a table shown on Figure 5.32 along with simulation, emulation and synthesis times for the three previously discussed use cases (general data).

Parameters Use Case		Use Cases Results					
		UC1		UC2		UC3	
Communication Scheme		CS1	CS2	CS1	CS2	CS1	CS2
General	Full sys. sim. time (hours)	33.3	0.12	48.3	1.2	66.6	66.6
	Full sys. synt. Time (min)	12	3	16	9	15	15
	DRNoC emul. Time (ms)	80	20	92	45	125	125
	DRNoC total emul. Time (min)	4		9		9	
	Worst Case sim. Time (min)	2.5	2	2.5	2.5	2.5	2.5
	Worst Case PR synt. Time (min)	10	7	10	3	10	0.5
Speedup	Total emulation to simulation Speedup = Emul./sim	501		247		888	
	PR synthesis to synthesis Speedup = PR synth./synth.	5.2	1.5	6.4	4	6	6
	PR emulation to synthesis Speedup = PR emul./synth.	144		281		400	

Figure 5.32: Emulation system evaluation for each use case presented in this section. Speedup comparison of the proposed, partial reconfiguration based approach, with respect to: i) simulation (Total emulation to simulation), ii) a full synthesis based approach (PR synthesis to synthesis) and iii) speedup of the worst case core synthesis time with respect to the full system synthesis time (PR emulation to synthesis).

The three main evaluation aspects that have been differentiated are described next:

1. **System emulation to system simulation speedup** (total emulation to simulation row in the table shown in Figure 5.32). This value gives the speedup which results from using emulation instead of simulation that is usually reported in the emulation systems related literature. Here, this value represents the sum of the time needed for the simulation of each system configuration for a given case (full

sys. sim. time row in the table shown in the figure), divided by the total time that is spent in each use case emulation (DRNoC total emul. time) that is in the minutes range.

It is worthy to remark that the total emulation time includes the data retrieving from the emulation system by the DRNoC emulation software running in a host PC and therefore this time is higher than the DRNoC FPGA emulation time, DRNoC mul. time row in the table, that is in the ms range, and, if the last time would be taken into account, then the achieve speedup is of four orders of magnitude.

2. *Partial reconfiguration synthesis time to full synthesis* (PR synthesis to synthesis row in the table shown in Figure 5.32). This value represents the speedup, which is gained with the proposed solution, as only a piece of the entire system has to be re-synthesized (when a hard core has to be included in the library). For this aim, the time needed for the entire systems synthesis (Full sys. synt. time in the table) has been compared to the worst case individually system elements synthesis time (Worst case synt. time in the table). The speedup achieved, compared to a no partially reconfigurable system, is in the range of several units.

3. *Partial reconfiguration emulation to full system synthesis* (PR emul. to synthesis row in the table shown in Figure 5.32). Gives the speedup of the partial reconfiguration emulation flow compared to the time spend on following the same steps but using full system synthesis that is the case of traditional emulation systems. For both approaches, the worst case has been assumed. For the proposed flow, the worst case is when all cores have to be loaded using partial reconfiguration and contrary, the worst case for the synthesis based approach, assuming that the emulation system is exactly the same, is when a new synthesis is required for each parameter to be changed.

For instance, for the use case 2 (UC2), a total of 16 seconds are required for all partial reconfigurations. Initially, 10 seconds are needed to load each slot/core, NI and router in the DRNoC FPGA that corresponds to CS1. Then, 1.5 seconds are required for the measuring point reconfiguration. After that, 3 seconds to pass to CS2 and finally 1.5 seconds more, again for the three needed measuring point reconfigurations. Differently, for the synthesis based approach, 3 re-synthesis of each configuration will be required, that is, 48 minutes for CS1. The same reasoning has been followed for UC3 and UC1.

The achieved speedup results are in the range of hundreds of times (being 400 times the best time reduction).

Nevertheless, for performing a more complex design space exploration, more configurations have to be synthesized and tested and, the achieved speedup will be much higher. Also, reconfiguration times can be drastically reduced if other programming interfaces are used, like the FPGA internal configuration port (ICAP) or the parallel programming port (SelectMAP).

However, the synthesis times reported in the use cases are relatively low (less than 20 minutes). Differently, for instance, in (Genko et al., 2005) it is said that each re-synthesis takes hours.

5.5.5.2 DRNoC Emulation: Advantages and Disadvantages

Table 5.17 summarizes some main advantages and disadvantages of the proposed emulation framework.

Table 5.17: Proposed partial reconfiguration base emulation framework advantages and disadvantages

Advantages	Disadvantages
Faster design space exploration	
Not restricted to NoCs	High area overhead
Low level core re-usability	More complex core design
Permits the use of more real models	Less observability than simulation

The main advantage is that the emulation framework permits to achieve *faster design space exploration* with respect to a no partially reconfigurable approach, as it has been reported by the speedups presented in the previous Table 5.32. Furthermore, it is *not restricted only to NoCs* allowing other communication schemes to be tested and, also, permits *high core re-usability at a lower level*, since a core included in the general hard core library can be directly used to build different systems. Another advantage is that the partial reconfiguration based approach permits to use more “real” models, because it reduces the need of having more logic grouped in a single model in order to have easy switching between a set of features. On the other hand the main drawbacks are mainly related to the partial reconfiguration technique available nowadays, that is, the high area overhead and the more complex cores design, as well as the reduced observability, when compared to a simulation approach, which has been reduced by partially reconfiguring different observations hardware (measuring points in this case).

5.6 Conclusions

Four original applications of partial reconfiguration, mostly (but not entirely) based on methods and tools proposed along the thesis have been presented. Each application conclusion can be found in the next paragraphs.

The first and most simple reconfigurable system targets to enhance the features of a wireless sensor network node and permit SW and/or HW updates. In this application, new configurations are transmitted using the wireless network. A reconfiguration use case has been defined to evaluate the possibility of using the network for new configuration transmissions. The reported results have shown that the cost of sending large amount of data through the network is high, more than the cost of reconfiguring the FPGA. Therefore it is highly recommended to put extra efforts on locally keeping as much hard cores as possible. The use of a partial runtime reconfigurable system has permitted to kept in the local memory some configurations and also, to retransmit them trough the network. This would have been impossible for this restricted device, if full reconfigurations had been used. Furthermore, to evaluate the cost of reconfiguration and remote updates, a set of parameters have been originally defined and can be further

used in the design of a reconfiguration control system. The reconfiguration process has been based on the JTAG standard and this makes the solution portable to other FPGA families. To summarize, the reconfigurable system, designed following some thesis contributions, has permitted to successfully exploit FPGA reconfigurability in a resource restricted wireless sensor node.

The second application has been a proof of concept related to the integration of reconfigurable systems in client-server environments. The entire environment has been oriented to deal with the client reconfigurability. The own client has been prepared to automatically and fully transparently receive and administrate configurations. A more flexible (compared to the WSN application) reconfigurable architecture has been required for this application, because the foreseen variety of hard cores to be consumed by the system was broad. As conclusion from this experience, it has to be said that the entire system works properly and the reconfigurable client has been successfully integrated in the client-server environment. The achieved reconfiguration process was fully transparent. However, some restrictions have been noticed. Some of these restrictions are related to the on-chip communications and others, to the required by some hard cores (MPEG) area. A solution to the problem of the on-chip communication poor flexibility has been proposed in Chapter 3. Furthermore, some problems related to the system stability that results from the software-like hard core treatment have been noticed. For solving this problem, a validation process (the third application) has been proposed.

The third application is related to an extension of the architecture used in the reconfigurable client. In this application a debug process, based on partial reconfigurations, along with a debug software has been used to test, debug and/or validate a hard core loaded in the reconfigurable system. The validation process and the system integration with the debug software have been successful. Probably in the future such debug and validation features might be obligatory for reconfigurable system.

The last application is a complete emulation framework. The proposed work flow reduces the systems design space exploration time and takes advantage of hard core reusability. The core of the framework is the exploitation of the DRNoC 2D reconfigurable system, proposed in Chapter 3. For designing hard cores to be included in the partial reconfiguration library, the design flow proposed in Chapter 4 has been used. Furthermore, the tools presented in the same chapter have been integrated in a graphical emulation control SW. Three framework use cases have been created and presented in section 5.5.4 to validate the work flow. The proposed emulation framework gives the possibility of low level core re-usability and permits fast design space exploration achieving speedups of up to 400X compared to a synthesis based solution (achieved speedups have been reported in section 5.5.5). On the other hand, the emulation system performance can be improved and extended by: i) changing the serial XUP board to host PC by an Ethernet link, ii) add more cores to the hard core library, iii) include more intelligence in the system and reduce the design flow manual steps and, iv) switch to another board with a bigger FPGA in order to map a bigger DRNoC architecture. Finally, the framework can be re-targeted to other FPGAs with the exception of the hard core library and the related hard cores manipulation tools that support Virtex II and Virtex II Pro FPGAs.

Based on the experience gained, at this point it could be said that the main problem

is not really related to finding the partial reconfiguration killer application domain, as it has been state in the Chapter abstract. All domains, including the presented in this thesis, could be a killer one. Unfortunately, the main deficiency that remains and does not permitted to say that the killer application has been found, is the area related problems. A lot of area is wasted due to partial reconfiguration and the no boundary crossing problem. This has not permitted to fully exploit reconfigurable systems and to demonstrate the applications real value. For instance the emulation framework would be more interesting if, an 8x8 DRNoC could have been used or if the MPEG codec was fully integrated in the remote reconfigurable system.

Chapter 6

Overall Conclusions and Future Work



Thesis summary, general conclusions and future work. Contributions summary and analysis.

This Chapter summarizes the main topics the thesis has been focused on. In the first section, 6.1, general conclusions of each Chapter have been included, while in section 6.2 a summary of the thesis contributions can be found. The related main publications have been included in section 6.3 and a short thesis achievements evaluation in section 6.4. Section 6.5 is oriented to future work and, section 6.6 discusses the lessons learned, answering some main questions which arose along the document.

6.1 Summary and General Conclusions

The architecture design method, the tools and the flows proposed along the thesis have followed the thesis main approach where the reconfigurable system and the hard core design processes are considered separately.

The work presented along the thesis gives a possible solution for designing flexible and scalable partial runtime reconfigurable systems based on commercial FPGAs. More specifically, in this thesis i) hard cores can be designed by designers with no special partial reconfiguration knowledge and even without knowing the target reconfigurable system details and ii) a reconfigurable system can consume a broader set of hard cores, even ones that are designed for another similar system.

Each aspect that involves the design of flexible and scalable reconfigurable systems has been discussed in a separate thesis Chapter. The overall conclusions related to each Chapter can be found next:

1. *In Chapter 2, a method for designing virtual architectures (a hardware abstraction that has been defined in the introduction Chapter 1) has been originally proposed.* Following the method general steps, one and two dimensional architectures for Xilinx Virtex II FPGAs have been defined in the same Chapter. In addition, two one dimensional architectures that target two specific Virtex II FPGA devices have been designed and discussed along the Chapter, as demonstrator example.

The 1D reconfigurable systems that can be designed using the proposed method have been compared with the state of the art. The comparison results presented demonstrate that, using the proposed method, more flexibility can be achieved. This flexibility is represented by allowing hard cores (partial configuration files) to be loaded in any slot position and, also, to be grouped to allocate bigger cores. Furthermore, more in detail, the bus based structures, used for the 1D reconfigurable systems have been compared with other buses. Due to the selected design approach, the designed buses occupy less FPGA area and have better performance, in terms of clock frequency, compared with other state of the art approaches.

2. Chapter 3 has been focused on reconfigurable system on-chip communication issues. In that Chapter, *a solution, where reconfigurability is extended to include not only the system processing cores, like most of the state of the art solutions including the presented in Chapter 2, but also to permit adaptability of the on-chip communications has been originally proposed.* The proposed solution, called Dynamic Reconfigurable NoC (DRNoC), covers several aspects. First, the DRNoC architecture defines the basics of the entire reconfigurable system flexibility and scalability. Differently from the state of the art, the DRNoC virtual architecture permits several communication strategies to coexist independently in the reconfigurable system. Furthermore, the architecture permits to define not only homogeneous communications, NoC and/or bus, like other approaches, but also to define heterogeneous communication schemes, where several strategies can be combined (similarly to hierarchical NoCs), point

to point, point to multipoint, bus and/or NoC. The architecture has been mapped to Virtex II FPGAs by adapting a virtual architecture selected from Chapter 2. Second, reconfiguration solutions that span through different levels of the OSI communication model have been proposed. Third, a set of design resources that permit to create NoC communication schemes have been presented, including an original dynamically reconfigurable router with separate data and control path (although the router practical implementation was not possible), along with an original solution for system addressing and NoC packet format. Additionally, a software tool that automatically generates NoC configurations has been presented.

The resulting reconfigurable system has been compared with other state of the art solutions. Comparison results have been based in some parameters, selected from a state of the art work and others that have been originally proposed in this thesis. From the presented results, it can be concluded that the proposed 2D architecture has the highest flexibility and integration. Furthermore, as it is based on a NoC communication, the proposed solution is highly scalable.

3. A total of three reconfigurable systems virtual architectures have been proposed in the first two Chapters. Two of them one dimensional, in Chapter 2, and one two dimensional in Chapter 3. To give support to the design of a complete reconfigurable system, based on the already presented highly flexible and scalable architectures and following the thesis approach, where the hard core design process and the hard core adaptation in the target system are considered separate processes, a set of tools and a design flow have been originally proposed in Chapter 4. The main objective of the proposed tools and the hard core design flow is to provide a solution where the details of the target system architecture are not known at design time and the target hard core position is not required. Furthermore, the proposed pBITPOS tool, which is intended to be integrated in embedded reconfigurable systems, permits to load a core designed for a Virtex II FPGA in a suitable position in a target Virtex II Pro virtual architecture. The provided tools, based on a direct bitstream manipulation solution for Virtex II and Virtex II Pro FPGAs, also originally proposed in the same Chapter, have been widely tested and performance results have been provided and analyzed. The presented tools features have been compared with state of the art works. The comparison results show that, BITPOS and pBITPOS share some of their features with other academic tools, but provide the widest flexibility.

The proposed hard core design flow, based on system and architecture templates and the BITPOS tool, has permitted to achieve a straightforward design flow where a single synthesis is needed in order to generate hard cores while, with the Xilinx approach, as many projects as possible reconfigurable system combinations have to be defined. Also, the design flow partially overcomes the requirement of being expert designer and permits non partial reconfiguration experts to design hard cores for reconfigurable systems. Some hardware designers with no partial reconfiguration knowledge, following the flow, have successfully designed several hard cores. However, designers, on the other hand, still have to manually check if there are wires crossing the defined slot boundaries. This restriction is directly derived from the place and route tools currently provided by Xilinx, and

might not be needed in the future.

To conclude, *this Chapter has provided a complete tool-set and a design flow that make possible a system to consume a broader set of hard cores (including ones designed for a similar system) on one hand and, on the other hand, hard core designers to be non partial reconfiguration experts that do not know the target systems specifics.* However, designers of virtual architectures for reconfigurable systems still have to be partial reconfiguration experts.

4. The applications *Chapter demonstrates some of the advantages of partial runtime reconfigurable systems by exploiting, in most of the cases, the flexibility and scalability provided by the proposed design method and tools.* Reconfigurable systems with different degree of flexibility have been designed and integrated in original application domains as proof of concept. *The first, and the simplest, system originally explores the possibility of sending new configurations, through a wireless network, to a highly restricted reconfigurable device.* The application domain in this case is a Wireless Sensor Network (WSN). A set of cost based evaluation parameters have been originally defined and used to demonstrate that partial reconfiguration is more suitable than full reconfigurations for these resource restricted devices due to its reduced cost (mainly memory and transmissions cost). On the other hand, the advantages of the architectures, method and tools proposed in Chapter 2, Chapter 3 and Chapter 4 have not been exploited in this application domain, as the node main task of sensing is well defined, and because the target Spartan 3 FPGA has reduced area and reconfiguration features. To conclude, one of the most costly activities in the industrial use of WSNs is the deployment phase, including the actual installation, programming and debugging of the nodes in a given space. By using systems like the one presented, and reprogramming the hardware via radio, it will be possible to re-use and create new applications on networks that have been already deployed.

Differently, the advantages of the thesis proposals have been exploited in the second application, where a reconfigurable multimedia device is integrated in a client-server environment. In such systems, hard cores to be loaded in the device are not known in advance and thus system flexibility and reusability is mandatory. *The method proposed in Chapter 2 along with the tools and flows proposed in Chapter 4 have led to the successful design and integration of the remote reconfigurable client.* On the other hand, due to the place and route restrictions of tools, currently provided by Xilinx, there is a lot of area wasted in order to guarantee that no wires cross slot boundaries. Therefore, it has not been possible to test the remote reconfigurable system with real live multimedia cores, like MPEG2 and MPEG4, although video was played using previous simpler form.

In the third application, the architecture used in the multimedia client has been extended to support the insertion of debug modules in the system. This, new architecture, has been integrated in a debug software (previously developed in the research center). This original approach for debugging has been successfully tested.

The last application is an original emulation approach where the benefits of flexible reconfigurable systems (architecture and support tools) have permitted

to exploit hard core reusability for fast system emulation. A complete emulation framework has been originally proposed, designed and tested along the Chapter. To demonstrate the feasibility of the proposed emulation method, three use cases have been arranged. A speedup analysis has been performed and results have shown that with the presented approach the emulation process can substantially be improved, being 400X the highest reported speedup, compared to a no partial reconfiguration flow, and 144X the lowest. Again, restrictions of the currently available reconfiguration technique have not permitted to test the system with real live applications.

6.2 Summary of Original Contributions

A summary list of the main original contributions presented and highlighted along the thesis document can be found below:

- A Virtual Architecture design method that targets the design of flexible and scalable reconfigurable systems. The method, firstly proposed in this thesis, summarizes the state of the art and the thesis knowledge, and proposes a set of steps that permit to design more flexible systems, in term of the place and diversity of the hard cores than can be loaded and also, more scalable systems, where several reconfigurable regions with different interconnectivity can be defined.
- An original on-chip reconfigurable communication solution based on adapted network on chip, where not only cores can be reconfigured, but the on-chip communications can be adapted using partial reconfiguration to different communication schemes. The solution, that provides the highest flexibility from the state of the art includes, the definition of the NoC-base communication architecture, the definition of the reconfiguration techniques, the system addressing and the design resources, and, more specific for the NoC communication schemes, an original packet format and router architecture intended for data path adaptability.
- A direct bitstream manipulation solution for Virtex II/Pro FPGAs that is the most complete one from the state of the art, along with a set of tools for hard core extraction and relocation that where the firstly available solutions for these FPGAs.
- An original partial runtime reconfigurable systems hard cores design flow based on architecture templates that permits to design hard cores by non partial reconfiguration experts, and without knowing the target reconfigurable system details (routing and other cores allocation).
- The first state of the art approach in designing and testing fine grain dynamic reconfigurable systems for nodes for wireless sensor networks, along with a set of evaluation parameters that permit to estimate the worthiness of a reconfiguration process.

- A setup for the integration of a reconfigurable system in a client-server environment that includes an original reconfiguration control middleware and an automatic reconfiguration flow.
- A solution for the use of partial reconfiguration for debug.
- An original fast on-chip communications emulation method and framework, that are originally based on the idea of hard core reusability through partial runtime reconfiguration.

6.3 Main Publications

Part of the thesis results can be found in several publications:

1. The slot definition along with the first bus based reconfigurable system has been published in: Yana E. Krasteva, Ana B. Jimeno, Eduardo de la Torre and Teresa Riesgo, "Flexible Core Reallocation for Virtex II Structures" in Proceedings of the IEEE Intl. Embedded Reconfigurable Systems and Architectures (ERSA'05), pp. 189-195, Las Vegas (USA), June 2005.
2. The general Virtual Architecture design method has been published in: Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo "Virtual Architectures for Partial Runtime Reconfigurable Systems. Application to Network on Chip based SoC Emulation", in Proceedings of IEEE Annual Conference of the IEEE Industrial Electronics Society (IECON'08), pp. 211-216, Orlando-Florida (USA), November 2008.
3. The tools and the design flow have been published in: i) the BITPOS tool in Yana E. Krasteva, Ana B. Jimeno, Eduardo de la Torre and Teresa Riesgo, "Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs", in Proceedings of the IEEE Intl. Workshop on Rapid System Prototyping (RSP'05) Montreal (Canada), pp. 77-83, June 2005 and ii) the pBITPOS tool in Yana E. Krasteva, Didier Joly, Eduardo de la Torre and Teresa Riesgo "Virtex II Bitstream Manipulation: Application to Reconfiguration Control Systems ", in Proceedings of 16th IEEE Intl. Conference on Field Programmable Logic and Applications (FPL'06), pp. 1-4, Madrid (Spain), August 2006.
4. The adaptive communication system main idea has been firstly published in Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo "Partial Reconfiguration for Core Reallocation and Flexible Communications", in International Workshop on Reconfigurable Communication-Centric System-on-Chips (ReCoSoC'06), pp. 91-97, Montpellier (France), July 2006. More details and some tests with preliminary results can be found in Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo "Reconfigurable Heterogeneous Communications and Core Reallocation for Dynamic HW Task Management" in Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'07), pp. 873-876, New Orleans (USA), May 2007.

5. The reconfigurable system applied to a wireless sensor network node has been published in: Yana E. Krasteva, Jorge Portilla, Jose M. Carnicer, Eduardo de la Torre and Teresa Riesgo "Remote HW-SW Reconfigurable Wireless Sensor Nodes", in Proceedings of IEEE Annual Conference of the IEEE Industrial Electronics Society (IECON'08), pp. 2483 - 2488, Orlando-Florida (USA), November 2008.
6. The remote reconfigurable ENAMORADO system has been published in the book chapter: Yana E. Krasteva; C. Papagianni ; E. Kosmatos; Eduardo de la Torre; I.S. Venieris; and Teresa Riesgo "ENAMORADO: Enabling Nomadic Agents in a Multimedia ORiented Architecture of Distributed Objects", in "Innovation and the Knowledge Economy", Edited by Paul Cunningham y Miriam Cunningham, IOS Press, pp. 975-982, October 2005.
7. The debug system basics have been published in: Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo, "Applying Partial Reconfiguration for Debugging and Monitoring FPGA based Reconfigurable Systems", in Proceedings of International Conference on Design of Circuits and Integrated Systems (DCIS'06), Barcelona (Spain), November 2006.
8. The application of the reconfigurable system to emulation, along with some preliminary results, has been published in: Yana E. Krasteva, Francisco Criado, Eduardo de la Torre and Teresa Riesgo, "A Fast Emulation-Based NoC Prototyping Framework", in International Conference on ReConFigurable Computing and FPGAs (ReConFig'08), pp. 212-216, IEEE 2008, Cancun (Mexico), December 2008.

A complete list of all the thesis publications can be found at the end of the document. Nevertheless, there are still some results, mainly related to the NoC based system and its application to emulation that have not been published so far.

6.4 Thesis Achievements

It is really difficult to evaluate the relevance of the work presented in this thesis. Probably the main contribution of the thesis work, related to enhancing partial reconfiguration research, has been to share the gained knowledge and ideas with other researchers. The tools presented in Chapter 4 have been provided to some research groups from different countries, the published papers with the thesis partial results have been referenced several times and, the experience in partial reconfiguration issues has been used to provide support to other researchers from different groups. Finally, this thesis work has successfully opened a new research line in the Centre of Industrial Electronics that has been, and is being, supported by the following projects:

- SMART (Secure, Mobile visual sensor networks ArchiTecture), ARTEMIS FP7 (ARTEMIS-2008-1- 100032 JTU), period: 2009-2011, partners: Hellenic Aerospace Industry (Coordinator), Thales Italia, Philips Consumer Lifetime, Metodos y Tecnologia (MTP), Lippert, Nanosens, Technische Universität Braunschweig

(Germany), Telecommunication Systems Institute (Greece) and Universidad Politécnica de Madrid (CEI). This Project involves the design of a complete dynamic reconfigurable system for a wireless sensor network node with high performance and low power characteristics.

- DR.SIMON (Dynamic Reconfigurability for Scalability In Multimedia Oriented Networks), Ministerio de Ciencia e Innovación (TEC2008-06486-C02-01), period: 2009-2011, partners: Universidad de las Palmas de Gran Canaria and Universidad Politécnica de Madrid (CEI). This project involves finding new scalable hardware solution (based on FPGAs dynamic reconfiguration) for running scalable video applications.
- ENDIVIA-DEVASAR (Depuración y Validación de SoCs basada en arquitecturas reconfigurables y NoCs), Ministerio de ciencia e Innovación, period: 2006-2008, partners: Universidad de las Palmas de Gran Canaria and Universidad Politécnica de Madrid (CEI).
- ENAMORADO (Enabling Nomadic Agents in a Multimedia-ORiented Architecture of Distributed Objects), IST (FP5), period: 2002-2005, partners: CANAL PLUS DELTATEC (Belgium), Siemens Mobile (Italy), MEMONDO Graphics (Spain), National Technical University of Athens (Greece) and Universidad Politécnica de Madrid (CEI).

To conclude, it could be said that the *thesis contributions have some importance, demonstrated by the amount of references some of the published papers have received, the tools and support requested by external researches and the fact that the laboratory work in this subject will be continued using some of the thesis contributions.*

6.5 Future Work

All the presented solutions have led to the establishment of a research line in the Center of Industrial Electronics of the Universidad Politécnica de Madrid that is going to be continued. Some of the possible and/or planed future works are briefly described next:

- In the applications presented in the thesis, the decision of when a certain reconfiguration process is triggered has been taken by: the remote server in the client-server application and by the network administration in the wireless sensor network. Differently, in the future work it is desired to provide more intelligence to the reconfigurable device, including the possibility of taking reconfiguration related decision, locally and/or distributed. This future work will be part of the previously mentioned SMART and DR. SIMON projects. For instance, in SMART, wireless sensor networks will be considered as a unique system, composed of several individuals that work together and adapt themselves and their neighbors to achieve a certain goal.
- In all the presented systems, the processing element was outside the FPGA. It will be interesting to test a tightly coupled system with an on-chip control in

charge of the entire reconfiguration process. This approach might be followed in both mentioned projects, but oriented to different applications. In DR.SIMON the Scalable Video Coding extension of the H.264 (H.264/SVC) video standard is targeted as we consider that this new standard is well suited for self-adaptive and scalable runtime reconfigurable systems with smaller-grain reconfiguration than the ones presented in this thesis. On the contrary, in the SMART project encryption algorithms (mainly Elliptic Curve Cryptography - ECC) and H.264/AVC (Advance Video coding) will be targeted. However, some of the thesis original contribution, like for instance the virtual architecture design method, will be used in these projects.

- The presented systems could be applied to other domains, like digital power control systems, evolvable hardware systems and bio-inspired technology, among others. Digital control is a main research line in the Center of Industrial Electronics and is well suited for the reconfigurable systems presented in the thesis as the required resources are not high, compared to other applications, like for instance, the video applications.
- The virtual architecture design method can be applied to Virtex 5 FPGAs in order to port the systems presented along the thesis to these devices.
- The DRNoC framework could be improved by: i) including more hard cores in the library and improving the SW in terms of performance and reliability, ii) also, the XUP board to the PC serial link could be substituted by a Ethernet link in order to overcome the reported problems with the buffers and iii) the non automated steps could be automated.

6.6 Discussion on Reconfigurable Systems

This section, in order to summarize, give answers to some questions that have appeared along the thesis work.

The first question is: *"is reconfiguration worthy?"*. It is difficult to estimate if reconfiguration, and more specifically partial reconfiguration, is worthy or not. Designing a complete reconfigurable system, nowadays, is not an easy task. It requires a very difficult and tedious process. Furthermore, the commercially available solutions are quite restrictive, complex to use and require specialized designers. The complexity starts from the basics of the reconfigurable systems, the hardware infrastructure, goes through the hard core design flows and methods and ends by the runtime reconfiguration control SW (some of these aspects have been addressed within the work presented in this thesis). However, the first, direct, answer to the question is; yes, sure, reconfiguration is worthy (the entire document is about reconfigurability). But this answer is not fully correct, the exact answer is that, reconfigurability, nowadays, is very promising, but difficult to execute and therefore it is too costly and it is not worthy for the industry. Differently, in research, these uncertainties are taken as opportunities and reconfiguration is worthy.

Another question, discussed along the thesis, was: *which is the partial reconfiguration killer application?*. In a look at the industry state of the art, it seems that the first partial runtime reconfigurable system in the market will be related to software defined radio or to cryptography. However, based on the gained experience, at this point, it could be said that the main problem is not related to the killer application domain because all of them, including those presented in this thesis and the planned in the future work (H.264/SCV for instance), could be the breakthrough to the industry. The main application related problem is the area wasted due to poor routing tools, restricted runtime support and long reconfiguration times that do not permit to test systems in real environments.

Complete solutions to the problems mentioned before are required and, as it has been stated in the thesis introduction, the research community in reconfigurable systems is pushing the industry by providing possible solutions and being several steps ahead of it. However, the improvements rely mainly on the reconfiguration technology provided by FPGA companies. From here, the next question arises: *will future devices be runtime reconfigurable?*. The answer to this question, without doubt, is affirmative. Not only partial reconfigurability of Xilinx FPGAs will be improved, but we believe that other FPGA providers will include this feature.

After several years of being the research community ahead, the main FPGA provider, the "de-facto" partially reconfigurable FPGAs provider (Xilinx), has changed its commercial policy and now they put special efforts on improving partial reconfiguration technology. In 2006, Xilinx has opened a research center focused on studying and improving partial reconfiguration technology. Furthermore, some clear improvements in the technology can be noticed in the newest FPGAs. The bitstream secrecy was high for former Virtex II and Virtex II Pro devices that appeared in 2002 - 2003 and a small part of the bitstream format (related to the addressing scheme) was released in 2005. Differently, for the newest Virtex 4 and Virtex 5, this data has been public along with the device appearance in the market. With the improving of the reconfiguration techniques and place and route tools, system solutions, like the presented in this thesis, will be applicable to real live. Regarding other FPGA providers, with the increased FPGA logic integration that results in extremely large FPGA configuration files and high configuration times (now reaching 20 MB and several tens of seconds) and, also, the increasing variety of available embedded cores, partial reconfiguration, probably, will be a default feature. This will result in more competitors in this area and will further improve the provided solutions.

Anyway, there is still a gap from the research topic to the industrial use. Therefore, there will be several years of research to have an easy to use and fully transparent reconfiguration solution, and make all applications suitable to show the benefits of partial runtime reconfigurable systems.

Bibliography

- [ABD92] J. M. Arnold, D. A. Buell, and E. G. Davis. Splash 2. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 316–322, 1992.
- [ACC⁺95] R. Amerson, R.J. Carter, W.B. Culbertson, P. Kuekes, and G. Snider. Teramac-configurable custom computing. *FPGAs for Custom Computing Machines, 1995. Proceedings. IEEE Symposium on*, pages 32–38, Apr 1995.
- [AH06] Bashir Al-Hashimi. *System-on-chip: Next Generation Electronics*. IEE, 2006.
- [Arn93] J.M Arnold. The splash 2 software environment. In IEEE, editor, *Proceedings. IEEE Workshop on FPGAs for Custom Computing Machines*, pages 88 – 93, April 1993.
- [Atm] Atmel. <http://www.atmel.com/>.
- [ATM05] ATMEL. *FPSLIC on-chip Partial Reconfiguration of the Embedded AT40K FPGA*. ATMEL, September 2005.
- [BA05] Christophe Bobda and Ali Ahmadinia. Dynamic interconnection of reconfigurable modules on reconfigurable devices. *IEEE Design & Test of Computers*, 22(5):443–451, 2005.
- [BBKG07] Frank Bouwens, Mladen Berekovic, Andreas Kanstein, and Georgi Gaydadjiev. Architectural exploration of the adres coarse-grained reconfigurable array. In Pedro C. Diniz, Eduardo Marques, Koen Bertels, Marcio Merino Fernandes, and João M. P. Cardoso, editors, *Reconfigurable Computing: Architectures, Tools and Applications, Third International Workshop, ARC 2007, Mangaratiba, Brazil, March 27-29, 2007*, volume 4419 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007.
- [BBP04] Sophie Bouchoux, El-Bay Bourennane, and Michel Paindavoine. Implementation of jpeg2000 arithmetic decoder using dynamic reconfiguration of fpga. In *ICIP*, pages 2841–2844, 2004.
- [BBSG08] Frank Bouwens, Mladen Berekovic, Bjorn De Sutter, and Georgi Gaydadjiev. Architecture enhancements for the adres coarse-grained reconfigurable array. In Per Stenström, Michel Dubois, Manolis Katevenis, Rajiv Gupta, and Theo Ungerer, editors, *High Performance Embedded Architectures and Compilers, Third International Conference, HiPEAC 2008, Göteborg, Sweden, January 27-29, 2008, Proceedings*, volume 4917 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 2008.
- [BJRK⁺03] Brandon Blodget, Philip James-Roxby, Eric Keller, Scott McMillan, and Prasanna Sundararajan. A self-reconfiguring platform. In *FPL*, pages 565–574, 2003.

- [bJVBR⁺96] by J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard. Active memories: Reconfigurable systems come of age. *IEEE Transactions on VLSI Systems*, 4(1):56–69, March 1996.
- [BM02] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *IEEE Computer*, 35(1):70–78, 2002.
- [BMA⁺05a] Christophe Bobda, Mateusz Majer, Ali Ahmadinia, Thomas Haller, André Linarth, and Jürgen Teich. The erlangen slot machine: Increasing flexibility in fpga-based reconfigurable platforms. In Gordon J. Brebner, Samarjit Chakraborty, and Weng-Fai Wong, editors, *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology, FPT 2005, 11-14 December 2005, Singagore*, pages 37–42. IEEE, 2005.
- [BMA⁺05b] Christophe Bobda, Mateusz Majer, Ali Ahmadinia, Thomas Haller, André Linarth, Jürgen Teich, Sándor P. Fekete, and Jan van der Veen. The erlangen slot machine: A highly flexible fpga-based reconfigurable platform. In *FCCM*, pages 319–320, 2005.
- [BMK⁺04] Christophe Bobda, Mateusz Majer, Dirk Koch, Ali Ahmadinia, and Jürgen Teich. A dynamic noc approach for communication in reconfigurable devices. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference , FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 1032–1036. Springer, 2004.
- [Bol03] I. Bolsens. Challenges and opportunities for fpga programmable system platforms. *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*, pages 3–, July 2003.
- [Bou06] Don Bouldin. Enhancing electronic systems using reconfigurable hardware. *IEEE Circuits & Devices Magazine*, 22(3):32–36, May 2006.
- [BP02] Kiran Bondalapati and Vicktor K. Prasanna. Reconfigurable computing systems. *Proceedings of IEEE*, 90(7):1201–1217, July 2002.
- [BR04] Gerard Habay Philippe Butel and Alain Rachet. *Managing partial dynamic reconfiguration in virtex-ii pro fpgas*. Xcell Journal Online, XILINX, 2004.
- [BT04] Krishnamoorthy Baskaran and Srikanthan Thambipillai. A hardware operating system based approach for run-time reconfigurable platform of embedded devices. *6th RTL Workshop*, Nov 2004.
- [Car09] José María Carnicer. Reconfigurabilidad parcial y dinamica en nodos de redes de sensores inalamblicas. Master's thesis, Escuela Técnica Superior de Ingenieros Industriales Universidad Politecnica de Madrid, Abril 2009.
- [CBN08] Fabio Garzia1 Claudio Brunelli and Jari Nurmi. A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities. *Journal of Real-Time Image Processing*, 3(1-2):21–32, March 2008.
- [CBvdV05] Mateusz Majer Jürgen Teich Sándor P. Fekete Christophe Bobda, Ali Ahmadinia and Jan van der Veen. Dynoc: A dynamic infrastructure for communication in dynamically reconfigurable devices. In Steven J. E. Wilton Tero Rissa and Philip Heng Wai Leong, editors, *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications (FPL), Tampere, Finland, August 24-26, 2005*, pages 153–158. IEEE, 2005.
- [CCBM04] E. Carvalho, N. Calazans, E. Briao, and F. Moraes. Padreh - a framework for the design and implementation of dynamically and partially reconfigurable systems. *Integrated Circuits and Systems Design, 2004. SBCCI 2004. 17th Symposium on*, pages 10–15, Sept. 2004.

- [CCG⁺04] Marcello Coppola, Stephane Curaba, Miltos D. Grammatikakis, Riccardo Locatelli, Giuseppe Maruccia, and Francesco Papariello. Occn: a noc modeling framework for design exploration. *Journal of Systems Architecture*, 50(2-3):129–163, 2004.
- [CH02] Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.
- [CL07] Chang-Seok Choi and Hanho Lee. A self-reconfigurable adaptive fir filter system on partial reconfiguration platform. *IEICE Transactions*, 90-D(12):1932–1938, 2007.
- [CP04] Jeremy Chan and Sri Parameswaran. Nocgen: a template based reuse methodology for networks on chip architecture. In *17th International Conference on VLSI Design*, pages 717–720, 2004.
- [CYS06] H. Chaoui, M.C.E. Yagoub, and P. Sicard. Fpga implementation of a fuzzy controller for neural network based adaptive control of a flexible joint with hard nonlinearities. *IEEE International Symposium on Industrial Electronics*, 4:3124–3129, July 2006.
- [CZMS07] Christopher Claus, Johannes Zeppenfeld, Florian Helmut Müller, and Walter Stechele. Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system. In Rudy Lauwereins and Jan Madsen, editors, *2007 Design, Automation and Test in Europe Conference and Exposition (DATE 2007)*, April 16–20, 2007, Nice, France, pages 498–503. ACM, 2007.
- [DeH94] A. DeHon. Dpga-coupled microprocessors: commodity ics for the early 21st century. *FPGAs for Custom Computing Machines*, 1994. *Proceedings. IEEE Workshop on*, pages 31–39, Apr 1994.
- [DFR⁺05] Alberto Donato, Fabrizio Ferrandi, Massimo Redaelli, Marco D. Santambrogio, and Donatella Sciuto. Caronte: A complete methodology for the implementation of partially dynamically self-reconfiguring systems on fpga platforms. In *13th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2005)*, 17–20 April 2005, Napa, CA, USA, *Proceedings*, pages 321–322. IEEE Computer Society, 2005.
- [DL04] Mike Peattie Davin Lim. *XAPP 290 (v1.0)0: Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulation*. Xilinx, 2004.
- [DL07] F. Zhao D. Lymberopoulos, N. B. Priyantha. mplatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In *in Proc. of the 5th IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN'07)*, pages pp. 128–137, Cambridge, Massachusetts, April. 2007.
- [dM] Politecnico di Milano. Dresd. <http://www.dresd.org/>.
- [DMC⁺06] André DeHon, Yury Markovskiy, Eylon Caspi, Michael Chu, Randy Huang, Stylianos Perissakis, Laura Pozzi, Joseph Yeh, and John Wawrzynek. Stream computations organized for reconfigurable execution. *Microprocessors and Microsystems*, 30(6):334–354, 2006.
- [DMN⁺08] J. Delorme, J. Martin, A. Nafkha, C. Moy, F. Clermidy, P. Leray, and J. Palicot. A fpga partial reconfiguration design approach for cognitive radio based on noc architecture. *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, pages 355–358, June 2008.
- [DN06] Design and Reuse News. Xilinx and isr technologies announce software defined radio kit supporting partial reconfiguration and sca-enabled soc. <http://www.design-reuse.com>, 2006.

- [DNAKN] Masood Dehyadgari, Mohsen Nickray, Ali Afzali-Kusha, and Zainalabedin Navabi. A new protocol stack model for network on chip. In *2006 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2006)*, 2-3 March 2006, Karlsruhe, Germany, pages 440–441. IEEE Computer Society.
- [DT01] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 684–689. ACM, 2001.
- [EAGEG07] Esam El-Araby, Ivan Gonzalez, and Tarek El-Ghazawi. Performance bounds of partial run-time reconfiguration in high-performance reconfigurable computing. In *HPRCTA '07: Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications*, pages 11–20, New York, NY, USA, 2007. ACM.
- [EEAEG07] Ivan Gonzalez Esam El-Araby and Tarek A. El-Ghazawi. Bringing high-performance reconfigurable computing to exact computations. In Koen Bertels, Walid A. Najjar, Arjan J. van Genderen, and Stamatis Vassiliadis, editors, *FPL 2007, International Conference on Field Programmable Logic and Applications, Amsterdam, The Netherlands, 27-29 August 2007*, pages 79–85. IEEE, 2007.
- [Ele08] Nikkei Electronics. <http://techon.nikkeibp.co.jp>, 2008.
- [ELH02] David E. Taylor Edson L. Horta, John W. Lockwood. Dynamic hardware plugins in an fpga with partial run-time reconfiguration. In IEEE, editor, *Proc. of the 39th Design Automation Conference*, pages 343–348. DAC 02, 2002.
- [Eto07] Emi Eto. *Difference-Based Partial Reconfiguration*. Xilinx, Dec-2007.
- [FBL01] M. Waldvogel F. Braun and J. Lockwood. Obiwan - an internet protocol router in reconfigurable hardware. Technical Report Tech. Rep. WUCS-01-11, Applied Research Lab. Department of Computer Science. Washington University, July 2001.
- [FM04] João Canas Ferreira and José Silva Matos. A development support system for applications that use dynamically reconfigurable hardware. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 886–890. Springer, 2004.
- [FS05] João Canas Ferreira and Miguel M. Silva. Run-time reconfiguration support for fpgas with embedded cpus: The hardware layer. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CA, USA*. IEEE Computer Society, 2005.
- [GADMB07] N. Genko, D. Atienza, G. De Micheli, and L. Benini. Feature - noc emulation: a tool and design flow for mpsoc. In *Volume 7, Issue 4, Circuits and Systems Magazine, IEEE*, pages 42 – 51, Fourth Quarter 2007.
- [GAM⁺05] Nicolas Genko, David Atienza, Giovanni De Micheli, Luca Benini, Jose Manuel Mendias, Román Hermida, and Francky Catthoor. A novel approach for network on chip emulation. In *ISCAS (3)*, pages 2365–2368, 2005.
- [GASF03] Manuel G. Gericota, Gustavo R. Alves, Miguel L. Silva, and José M. Ferreira. Run-time management of logic resources on reconfigurable systems. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10974–10979. IEEE Computer Society, 2003.
- [GdITAR04] M. Garcia, E. de la Torre, F. Ariza, and T. Riesgo. Hardware and software debugging of fpga based microprocessor system through debug logic insertion. In *in Proc. of*

- the 14th Field-Programmable Logic and Applications (FPL'04)*, August 30-September 1 2004.
- [GDS06] By Mark Goosman, Nij Dorairaj, and Eric Shiflet. *How to Take Advantage of Partial Reconfiguration in FPGA Designs*. SoC Central - Electronic Edition, September 2006.
- [GG00] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *2000 Design, Automation and Test in Europe (DATE 2000)*, 27-30 March 2000, Paris, France, pages 250–256. IEEE Computer Society, 2000.
- [GIP⁺07] Cristian Grecu, André Ivanov, Partha Pratim Pande, Axel Jantsch, Erno Salminen, Ümit Ogras, and Radu Marculescu. Towards open network-on-chip benchmarks. In *First International Symposium on Networks-on-Chips, NOCS 2007*, 7-9 May 2007, Princeton, New Jersey, USA, *Proceedings*, page 205. IEEE Computer Society, 2007.
- [Gro08] Hipeac Group, 2008. <http://www.hipeac.net/>.
- [GZ97] Rajesh K. Gupta and Yervant Zorian. Introducing core-based system design. *IEEE Design & Test of Computers*, 14(4):15–25, 1997.
- [Har06a] Reiner Hartenstein. Why europe needs reconfigurable computing. *TU Kaiserslautern*, 2006.
- [Har06b] Reiner Hartenstein. Proposal pf a new jurnal by reneir hartenstein. In *TU Kaiserslautern*, Nov. 2006.
- [Har07] Reiner Hartenstein. The von neumann syndrome. In *invited paper Symposium The Future of Computing*, Delft Sept. 28 2007.
- [HCG07] Andreas Hansson, Martijn Coenen, and Kees Goossens. Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip. In Rudy Lauwereins and Jan Madsen, editors, *2007 Design, Automation and Test in Europe Conference and Exposition (DATE 2007)*, April 16-20, 2007, Nice, France, pages 954–959. ACM, 2007.
- [HDA⁺02] Schmit H., Whelihan D., Tsai A., Moe M., Levine B., and Reed Taylor R. Piperench: A virtualized programmable datapath in 0.18 micron technology. *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 63–66, 2002.
- [HFHK04] S. Hauck, T.W. Fry, M.M. Hosler, and J.P. Kao. The chimaera reconfigurable functional unit. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(2):206–217, Feb. 2004.
- [HG07] Thomas Hollstein and Manfred Glesner. Advanced hardware/software co-design on reconfigurable network-on-chip based hyper-platforms. *Computers & Electrical Engineering*, 33(4):310–319, 2007.
- [HL04] Edson L. Horta and John W. Lockwood. Automated method to generate bitstream intellectual property cores for virtex fpgas. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference , FPL 2004*, Leuven, Belgium, August 30-September 1, 2004, *Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 975–979. Springer, 2004.
- [HLK] Edson L. Horta, John W. Lockwood, and Sergio Takeo Kofuji. Using parbit to implement partial run-time reconfigurable systems. In Manfred Glesner, Peter Zipf, and Michel Renovell, editors, *Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream, 12th International Conference, FPL 2002*, Montpellier, France, September 2-4, 2002, *Proceedings*, volume 2438 of *Lecture Notes in Computer Science*, pages 182–191. Springer.

- [HLZ⁺06] Thomas Hollstein, Ralf Ludewig, Heiko Zimmer, Christoph Mager, Simon Hohenstern, and Manfred Glesner. Hinoc: A hierarchical generic approach for on-chip communication, testing and debugging of socs. *VLSI-SOC: From Systems to Chips*, 200/200(4):39–54, 2006.
- [HP03] J. Hennessey and D. Patterson. *Computer Architectures: A Quantitative Approach*. Morgan Kaufman, 2003.
- [HZG06] H. Hinkelmann, P. Zipf, and M. Glesner. Design concepts for a dynamically reconfigurable wireless sensor node. *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, pages 436–441, June 2006.
- [IFAC07] Tommaso Melodia Ian F. Akyildiz and Kaushik R. Chowdhury. A survey on wireless multimedia sensor networks. *Comput. Netw.*, 51(4):921–960, 2007.
- [inc03] Xilinx inc. *Xilinx application notes, XAPP 151 v1.6 :Virtex Series Configuration Architecture User Guide*, March 2003.
- [inc07] Xilinx inc. *Xilinx application notes, XAPP 058 v0.4 : In-System Programming Using an Embedded Microcontroller*, October 2007.
- [ins06] instat. <http://www.instat.com>, 2006.
- [ITR03] ITRS. *INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS 2003 - DESIGN*. 2003.
- [JBH] Adam Donlin Jürgen Becker and Michael Hübner. New tool support and architectures in adaptive reconfigurable computing. In *IFIP VLSI-SoC 2007, IFIP WG 10.5 International Conference on Very Large Scale Integration of System-on-Chip, Atlanta, GA, USA, 15-17 October 2007*, pages 134–139. IEEE.
- [JBHC⁺07] Michael Hübner Jürgen Becker, Gerhard Hettich, Rainer Costapel, Joachim Eisenmann, and Jürgen Luka. *Dynamic and Partial FPGA Exploitation*, volume 95 of 2. Proceedings of IEEE, February 2007.
- [JBP06] Michael Hübner Jürgen Becker and Katarina Paulsson. Physical 2d morphware and power reduction methods for everyone. In Gordon J. Brebner Peter M. Athanas, Jürgen Becker and Jürgen Teich, editors, *Dynamically Reconfigurable Architectures*, volume 06141 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
- [JFBCR⁺08] Jaume Joven, Oriol Font-Bach, David Castells-Rufas, Ricardo Martinez, Lluís Teres, and Jordi Carrabina. xenoc - an experimental network-on-chip environment for parallel distributed computing on noc-based mpsoc architectures. In *16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), 13-15 February 2008, Toulouse, France*, pages 141–148. IEEE Computer Society, 2008.
- [JMBM04] Antoine Jalabert, Srinivasan Murali, Luca Benini, and Giovanni De Micheli. xpipesCompiler: A tool for instantiating application specific Networks on Chip. In *Design, Automation and Test in Europe (DATE)*, Paris, France, February 2004.
- [JRG00] Philip James-Roxby and Steven A. Guccione. Automated extraction of run-time parameterizable cores from programmable device configurations. In *8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000), 17-19 April 2000, Napa Valley, CA, Proceedings*, pages 153–164. IEEE Computer Society, 2000.
- [JWLT00] Jonathan S. Turner John W. Lockwood and David E. Taylor. Field programmable port extender (fpx) for distributed routing and queuing. In *FPGA*, pages 137–144, 2000.

- [JWLZ03] C. Neely J. W. Lockwood and C. Zuver. An extensible, system-on-programmable-chip, content-aware internet firewall. In IEEE, editor, *Field-Programmable Logic and Applications*, pages 859–868. FPL, Sept 2003.
- [Kao05] Cindy Kao. *Benefits of Partial Reconfiguration*. Xilinx, Inc, 2005.
- [KBD07] Jiri Kadlec, Roman Bartosinski, and Martin Danek. Accelerating microblaze floating point operations. In Koen Bertels, Walid A. Najjar, Arjan J. van Genderen, and Stamatias Vassiliadis, editors, *FPL 2007, International Conference on Field Programmable Logic and Applications, Amsterdam, The Netherlands, 27-29 August 2007*, pages 621–624. IEEE, 2007.
- [KdlTRJ06] Yana Esteves Krasteva, Eduardo de la Torre, Teresa Riesgo, and Didier Joly. Virtex ii fpga bitstream manipulation: Application to reconfiguration control systems. In *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, August 28-30, 2006*, pages 1–4. IEEE, 2006.
- [KJdlTR05] Yana Esteves Krasteva, Ana B. Jimeno, Eduardo de la Torre, and Teresa Riesgo. Flexible core reallocation for virtex ii structures. In Toomas P. Plaks, editor, *Proceedings of The 2005 International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA 2005, Las Vegas, Nevada, USA, June 27-30, 2005*, pages 189–195. CSREA Press, 2005.
- [KJM⁺] Shashi Kumar, Axel Jantsch, Mikael Millberg, Johnny Öberg, Juha-Pekka Soininen, Martti Forsell, Kari Tiensyrjä, and Ahmed Hemani. A network on chip architecture and design methodology. In *2002 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2002), 25-26 April 2002, Pittsburgh, PA, USA*, pages 117–124. IEEE Computer Society, 2002.
- [KMK07] K. Kosciuszkiewicz, F. Morgan, and K. Kepa. Run-time management of reconfigurable hardware tasks using embedded linux. *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, pages 209–215, Dec. 2007.
- [KP06] Heiko Kalte and Mario Porrmann. Replica2pro: task relocation by bitstream manipulation in virtex-ii/pro fpgas. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 403–412, New York, NY, USA, 2006. ACM.
- [KPG07] Jürgen Becker Jean-Marc Philippe Katarina Paulsson, Michael Hübner and Christian Gamrat. On-line routing of reconfigurable functions for future self-adaptive systems - investigations within the æther project. In Koen Bertels, Walid A. Najjar, Arjan J. van Genderen, and Stamatias Vassiliadis, editors, *FPL 2007, International Conference on Field Programmable Logic and Applications, Amsterdam, The Netherlands, 27-29 August 2007*, pages 415–422. IEEE, 2007.
- [Lat] Lattice. <http://www.latticesemiconductor.com/>.
- [LBM⁺06] Patrick Lysaght, Brandon Blodget, Jeff Mason, Jay Young, and Brendan Bridgford. Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas. In *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, August 28-30, 2006*, pages 1–6. IEEE, 2006.
- [lib05] Dynamic and partial reconfiguration in fpga socs: Requirements tools and a case study. *New Algorithms, Architectures and Applications for Reconfigurable Computing - Springer US*, pages 157–168, 2005.
- [LMGVE04] Michael G. Lorenz, Luis Mengibar, Mario García-Valderas, and Luis Entrena. Power consumption reduction through dynamic reconfiguration. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference , FPL 2004, Leuven, Belgium, August*

- 30-September 1, 2004, *Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 751–760. Springer, 2004.
- [LMME07] Michael G. Lorenz, Luis Mengibar, Enrique San Millán, and Luis Entrena. Low power data processing system with self-reconfigurable architecture. *Journal of Systems Architecture*, 53(9):568–576, 2007.
- [MA07] Saeidi Samira Khademzadeh Ahmad Mehran and Armin. Smap: An intelligent mapping tool for network on chip. *Signals, Circuits and Systems ISSCS 2007*, pages 1–4, 13-14 July 2007.
- [MABT06] Mateusz Majer, Ali Ahmadiania, Christophe Bobda, and Jürgen Teich. A flexible reconfiguration manager for the erlangen slot machine. In *ARCS Workshops*, pages 183–194, 2006.
- [Mar06] Ivan Gonzalez Martinez. *Coprocesadores Dinámicamente Reconfigurables en Sistemas Embebidos basados en FPGAs*. PhD thesis, Universidad Autonoma de Madrid, March 2006.
- [Mat] MathStar. <http://www.mathstar.com/>.
- [MCM⁺04a] Leandro Möller, Ney Laert Vilar Calazans, Fernando Gehm Moraes, Eduardo Wenzel Brião, Ewerson Carvalho, and Daniel Camozzato. Fipre: An implementation model to enable self-reconfigurable applications. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 1042–1046. Springer, 2004.
- [MCM⁺04b] Fernando Gehm Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration*, 38(1):69–93, 2004.
- [MDN⁺07] T. Massey, F. Dabiri, H. Noshadi, P. Brisk, W. Kaiser, and M. Sarrafzadeh. Towards reconfigurable embedded medical systems. *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on*, pages 178–180, June 2007.
- [MDP02] Christian Plessl Matthias Dyer and Marco Platzner. Partially reconfigurable cores for xilinx virtex. In Manfred Glesner, Peter Zipf, and Michel Renovell, editors, *Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream, 12th International Conference, FPL 2002, Montpellier, France, September 2-4, 2002, Proceedings*, volume 2438 of *Lecture Notes in Computer Science*, pages 292–301. Springer, 2002.
- [MGC⁺07] Leandro Moller, Ismael Grehs, Ewerson Carvalho, Rafael Soares, Ney Calazans, and Fernando Moraes. A noc-based infrastructure to enable dynamic self reconfigurable systems. In Gilles Sassatelli, Manfred Glesner, Christophe Bobda, and Pascal Benoit, editors, *Proceedings of the 3rd International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2007, Montpellier, France, June 2007*, pages 23–30. Univ. Montpellier II, 2007.
- [MGCM06] Leandro Möller, Ismael Grehs, Ney Calazans, and Fernando Moraes. Reconfigurable systems enabled by a network-on-chip. In *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, August 28-30, 2006*, pages 1–4. IEEE, 2006.
- [MH06] Riad Ben Mouhoub and Omar Hammami. Noc monitoring hardware support for fast noc design space exploration and potential noc partial dynamic reconfiguration. *Industrial Embedded Systems, 2006. IES '06. International Symposium on*, pages 1–10, Oct. 2006.

- [MHB] Matthias Kühnle Michael Hübner, Christian Schuck and Jürgen Becker. New 2-dimensional partial dynamic reconfiguration techniques for real-time adaptive microelectronic circuits. In *2006 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2006)*, 2-3 March 2006, Karlsruhe, Germany, pages 97–102. IEEE Computer Society.
- [MHB04a] Lars Braun A. Klausmann Michael Hübner, Michael Ullmann and Jürgen Becker. Scalable application-dependent network on chip adaptivity for dynamical reconfigurable real-time systems. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference , FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 1037–1041. Springer, 2004.
- [MHB04b] Tobias Becker Michael Hübner and Jürgen Becker. Real-time lut-based network topologies for dynamic and partial fpga self-reconfiguration. In Edna Natividade da Silva Barros, Flávio Rech Wagner, Luigi Carro, and Franz-Josef Rammig, editors, *Proceedings of the 17th Annual Symposium on Integrated Circuits and Systems Design, SBCCI 2004, Pernambuco, Brazil, September 7-11, 2004*, pages 28–32. ACM, 2004.
- [MHB05] Katarina Paulsson Michael Hübner and Jürgen Becker. Parallel and flexible multiprocessor system-on-chip for adaptive automotive applications based on xilinx microblaze soft-cores. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CA, USA. IEEE Computer Society, 2005.
- [MHB06] Christian Schuck Michael Hübner and Jürgen Becker. Elementary block based 2-dimensional dynamic and partial reconfiguration for virtex-ii fpgas. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Proceedings, 25-29 April 2006, Rhodes Island, Greece. IEEE, 2006.
- [MHB08] Diana Göhringer Michael Hübner, Lars Braun and Jürgen Becker. Run-time reconfigurable adaptive multilayer network-on-chip for fpga-based systems. In *22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008*, pages 1–6. IEEE, 2008.
- [Miy98] T. Miyazaki. Reconfigurable systems: a survey. *Design Automation Conference 1998. Proceedings of the ASP-DAC '98. Asia and South Pacific*, pages 447–452, Feb 1998.
- [MM04] Srinivasan Murali and Giovanni De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 914–919, New York, NY, USA, 2004. ACM.
- [MMCM05] Aline Mello, Leandro Möller, Ney Calazans, and Fernando Gehm Moraes. Multinoc: A multiprocessing system enabled by a network on chip. In *2005 Design, Automation and Test in Europe Conference and Exposition (DATE 2005)*, 7-11 March 2005, Munich, Germany, pages 234–239. IEEE Computer Society, 2005.
- [MMP⁺] Daniel Mesquita, Fernando Gehm Moraes, José Palma, Leandro Möller, and Ney Laert Vilar Calazans. Remote and partial reconfiguration of fpgas: Tools and trends. In *17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings, page 177. IEEE Computer Society.
- [MMT05] Ali Ahmadiania Mateusz Majer, Christophe Bobda and Jürgen Teich. Packet routing in dynamically changing networks on chip. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CA, USA. IEEE Computer Society, 2005.

- [MNC⁺03] Jean-Yves Mignolet, Vincent Nollet, Paul Coene, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003)*, 3-7 March 2003, Munich, Germany, pages 10986–10993. IEEE Computer Society, 2003.
- [MNM⁺04] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. Bartic, W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Run-time support for heterogeneous multitasking on reconfigurable socs. *Integr. VLSI J.*, 38(1):107–130, 2004.
- [MNT⁺04] Mikael Millberg, Erland Nilsson, Rikard Thid, Shashi Kumar, and Axel Jantsch. The nostrum backbone - a communication protocol stack for networks on chip. In *17th International Conference on VLSI Design (VLSI Design 2004), with the 3rd International Conference on Embedded Systems Design*, 5-9 January 2004, Mumbai, India, pages 693–696. IEEE Computer Society, 2004.
- [MPE] MPEG21. <http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm>.
- [MSC⁺06] Leandro Möller, Rafael Soares, Ewerson Carvalho, Ismael Grehs, Ney Calazans, and Fernando Moraes. Infrastructure for dynamic reconfigurable systems: choices and trade-offs. In *SBCCI '06: Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pages 44–49, New York, NY, USA, 2006. ACM.
- [MSea97] William H Mangione-Smith and et al. Seeking solutions in configurable computing. *IEEE Computer*, 50(12):38–43, Aug 1997.
- [MTAB07] Mateusz Majer, Jürgen Teich, Ali Ahmadinia, and Christophe Bobda. The erlangen slot machine: A dynamically reconfigurable fpga-based computer. *VLSI Signal Processing*, 47(1):15–31, 2007.
- [MUB04] Björn Grimm Michael Ullmann, Michael Hübner and Jürgen Becker. An fpga run-time system for dynamical on-demand reconfiguration. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings*, 26-30 April 2004, Santa Fe, New Mexico, USA. IEEE Computer Society, 2004.
- [MVM06] S. Mahadevan, K. Virk, and J. Madsen. Arts: A systemc-based framework for modelling multiprocessor systems-on-chip. *Design Automation of Embedded Systems*, 2006.
- [NAS08] Christopher Claus Nicolas Alt and Walter Stechele. Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments. In *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*, pages 176–181. IEEE, 2008.
- [NCV⁺03] Vincent Nollet, Paul Coene, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. Designing an operating system for a heterogeneous reconfigurable soc. In *IPDPS*, page 7, 2003.
- [NDG05] Eric Shiflet Nij Dorairaj and Mark Goosman. *PlanAhead Software as a Platform for Partial Reconfiguration*. Xilinx, Inc, 2005.
- [NMV⁺04] Vincent Nollet, Théodore Marescaux, Diederik Verkest, Jean-Yves Mignolet, and Serge Vernalde. Operating-system controlled network on chip. In Sharad Malik, Limor Fix, and Andrew B. Kahng, editors, *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, pages 256–259. ACM, 2004.
- [OBD⁺05] B. O’Flynn, S. Bellis, K. Delaney, J. Barton, S.C. O’Mathuna, A.M. Barroso, J. Benson, U. Roedig, and C. Sreenan. The development of a novel minaturized modular

- platform for wireless sensor networks. *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 370–375, April 2005.
- [OMFK08] B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski. Socwire: A network-on-chip approach for reconfigurable system-on-chip designs in space applications. *Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, pages 51–56, June 2008.
- [OML⁺07] Ümit Y. Ogras, Radu Marculescu, Hyung Gyu Lee, Puru Choudhary, Diana Marculescu, Michael Kaufman, and Peter Nelson. Challenges and promising results in noc prototyping using fpgas. *IEEE Micro*, 27(5):86–95, 2007.
- [OMP⁺] Luciano Ost, Aline Mello, José Palma, Fernando Gehm Moraes, and Ney Calazans. Maia: a framework for networks on chip generation and verification. In Ting-Ao Tang, editor, *Proceedings of the 2005 Conference on Asia South Pacific Design Automation, ASP-DAC 2005, Shanghai, China, January 18-21, 2005*, pages 49–52. ACM Press.
- [OSM05] Tim Oliver, Bertil Schmidt, and Douglas Maskell. Hyper customized processors for bio-sequence database scanning on fpgas. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 229–237, New York, NY, USA, 2005. ACM.
- [PBdCR06] J. Portilla, J.L. Buron, A. de Castro, and T. Riesgo. A hardware library for sensors/actuators interfaces in sensor networks. In *Proc. of the 13th IEEE International Conference on Circuits and Systems*, Dec. 2006.
- [PBV07] Elena Moscu Panainte, Koen Bertels, and Stamatis Vassiliadis. The molen compiler for reconfigurable processors. *ACM Trans. Embedded Comput. Syst.*, 6(1), 2007.
- [PdCdITR06] J. Portilla, A. de Castro, E. de la Torre, and T. Riesgo. A modular architecture for nodes in wireless sensor networks. In *Journal of Universal Computer Science (JUCS)*, vol. 12, n^o 3, pages 328–339, March 2006.
- [PdMM⁺02] Jossé Carlos Palma, Aline Vieira de Mello, Leandro Moller, Fernando Moraes, and Ney Calazans. Core communication interface for fpgas. In *SBCCI '02: Proceedings of the 15th symposium on Integrated circuits and systems design*, page 183, Washington, DC, USA, 2002. IEEE Computer Society.
- [PGJ⁺05a] Partha Pratim Pande, Cristian Grecu, Michael Jones, André Ivanov, and Resve Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEE Transactions on Computers*, 54(8):1025–1040, August 2005.
- [PGJ⁺05b] Partha Pratim Pande, Cristian Grecu, Michael Jones, André Ivanov, and Resve A. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Trans. Computers*, 54(8):1025–1040, 2005.
- [PHP⁺04] Alexandra Poetter, Jesse Hunter, Cameron Patterson, Peter M. Athanas, Brent E. Nelson, and Neil Steiner. Jhdlbits: The merging of two worlds. In *FPL*, pages 414–423, 2004.
- [PJ06] John Lockwood Phillip Jones, Young Cho. An adaptive frequency control method using thermal feedback for reconfigurable hardware applications. In IEEE, editor, *IEEE International Conference on Field Programmable Technology*, Dec 2006.
- [QTSN08] Yang Qu, Kari Tiensyrja, Juha-Pekka Soininen, and Jari Nurmi. Design and low instantiation for run-time reconfigurable systems: A case study. *EURASIP Journal on Embedded Systems*, pages 1–9, 2008.

- [RAMV07] Juan J. Rodriguez-Andina, Maria J. Moure, and Maria D. Valdes. Features, design tools, and application domain of fpgas. *IEEE Trans. on Industrial Electronics*, 54(4):1810 – 1823, Aug 2007.
- [RS02] Anup Kumar Raghavan and Peter Sutton. Jpg - a partial bitstream generation tool to support partial reconfiguration in virtex fpgas. In *16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 15-19 April 2002, Fort Lauderdale, FL, USA, CD-ROM/Abstracts Proceedings. IEEE Computer Society, 2002.
- [RSMC07] Frans Trouw R Scrofano, Maya B. Gokhale, Virktor K. Eric Monmasson, and Marcian N. Cirstea. Fpga design methodology for industrial control systems - a review. *IEEE Trans. on Industrial Electronics*, 54(4):1824 – 1842, Aug 2007.
- [RSMC08] Frans Trouw R Scrofano, Maya B. Gokhale, Virktor K. Eric Monmasson, and Marcian N. Cirstea. Accelerating molecular dynamics simulations with reconfigurable computers. *IEEE Trans. on Parallel and Distributed Systems*, 19(6):764–778, June 2008.
- [SCK07] Krishnan Srinivasan, Karam S. Chatha, and Goran Konjevod. Application specific network-on-chip design with guaranteed quality approximation algorithms. In *Proceedings of the 12th Conference on Asia South Pacific Design Automation, ASP-DAC 2007, Yokohama, Japan, January 23-26, 2007*, pages 184–190. IEEE, 2007.
- [Sek03] Lukáš Sekanina. Towards evolvable ip cores for fpgas. In *EH '03: Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, page 145, Washington, DC, USA, 2003. IEEE Computer Society.
- [Sek07] Lukáš Sekanina. Evolutionary functional recovery in virtual reconfigurable circuits. *JETC*, 3(2), 2007.
- [Sek08] Lukáš Sekanina. *Evolvable Components*, volume 9783540403777. Springer-Verlag New York, LLC, September 2008.
- [SF06a] Miguel L. Silva and João Canas Ferreira. Support for partial run-time reconfiguration of platform fpgas. *Journal of Systems Architecture*, 52(12):709–726, 2006.
- [SF06b] Miguel L. Silva and João Canas Ferreira. Support for partial run-time reconfiguration of platform fpgas. *Journal of Systems Architecture*, 52(12):709–726, December 2006.
- [SKH08] Erno Salminen, Ari Kulmala, and Timo D. Hamalainen. Survey of network-on-chip proposals. WHITE PAPER, OCP-IP, MARCH 2008.
- [SKW⁺07] Gerard J. M. Smit, André B. J. Kokkeler, Pascal T. Wolkotte, Philip K. F. Hölzenspies, Marcel D. van de Burgwal, and Paul M. Heysters. The chameleon architecture for streaming dsp applications. *EURASIP J. Embedded Syst.*, 2007(1):11–11, 2007.
- [SLL⁺00] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, and Eliseu M. Chaves Filho. *orphoSys*: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Computers*, 49(5):465–481, 2000.
- [SMAA06] A. Susu, M. Magno, A. Acquaviva, and D. Atienza. Reconfiguration strategies for environmentally powered devices: Theoretical analysis and experimental validation. *Transactions on HiPEAC I*, pages 341–360, June 2006.
- [SS06] Marco D. Santambrogio and Donatella Sciuto. Partial dynamic reconfiguration: The caronte approach. a new degree of freedom in the hw/sw codesign. In *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL)*, Madrid, Spain, August 28-30, 2006, pages 1–2. IEEE, 2006.

- [TMG⁺05] Leonel Tedesco, Aline Mello, Diego Garibotti, Ney Calazans, and Fernando Moraes. Traffic generation and performance evaluation for mesh-based nocs. In Carlos Galup-Montoro, Sergio Bampi, and Alex Orailoglu, editors, *Proceedings of the 18th Annual Symposium on Integrated Circuits and Systems Design, SBCCI 2005, Florianopolis, Brazil, September 4-7, 2005*, pages 184–189. ACM, 2005.
- [TPA06] Roman Koch Thilo Pionteck and Carsten Albrecht. Applying partial reconfiguration to networks-on-chips. In *Proceedings of the 2006 International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, August 28-30, 2006*, pages 1–6. IEEE, 2006.
- [TPB07] Roman Koch Erik Maehle Michael Hübner Thilo Pionteck, Carsten Albrecht and Jürgen Becker. Communication architectures for dynamically reconfigurable fpga designs. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA*, pages 1–8. IEEE, 2007.
- [TPG] Thorsten Staake Thilo Pionteck, Thomas Stiefmeier and Manfred Glesner. On the design of a dynamically reconfigurable function-unit for error detection and correction. In Ricardo Augusto da Luz Reis, Adam Osseiran, and Hans-Jörg Pfleiderer, editors, *VLSI-SoC: From Systems To Silicon, Proceedings of IFIP TC 10, WG 10.5, Thirteenth International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC 2005), October 17-19, 2005, Perth, Australia*, volume 240 of *IFIP*, pages 283–297. Springer.
- [TPG04] Thomas Stiefmeier Lukusa D. Kabulepa Thilo Pionteck, Thorsten Staake and Manfred Glesner. On the design of a function-specific reconfigurable: hardware accelerator for the mac-layer in wlans. In Russell Tessier and Herman Schmit, editors, *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, FPGA 2004, Monterey, California, USA, February 22-24, 2004*, page 258. ACM, 2004.
- [TPK06] Carsten Albrecht Thilo Pionteck and Roman Koch. A dynamically reconfigurable packet-switched network-on-chip. In Georges G. E. Gielen, editor, *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6-10, 2006*, pages 136–137. European Design and Automation Association, Leuven, Belgium, 2006.
- [TTLH02] David E. Taylor, Jonathan S. Turner, John W. Lockwood, and Edson L. Horta. Dynamic hardware plugins: exploiting reconfigurable hardware for high-performance programmable routers. *Computer Networks*, 38(3):295–310, 2002.
- [Upe06] Andres Upegi. *Dynamically Reconfigurable Bio-inspired Hardware*. PhD thesis, ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE, March 2006.
- [VCC00] VCC. <http://www.vcc.com/prod1.html>, 2000.
- [VISI06] Jozsef Vasarhelyi, Maria Imecs, Csaba Szabo, and Ioan Iov Incze. Run-time reconfiguration of tandem inverter for induction motor drives. *Power Electronics and Motion Control Conference, 2006. EPE-PEMC 2006. 12th International*, pages 408–413, Aug. 2006.
- [VM04] Girish Varatkar and Radu Marculescu. On-chip traffic modeling and synthesis for mpeg-2 video applications. *IEEE Trans. VLSI Syst.*, 12(1):108–119, 2004.
- [VWG⁺04] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E.M. Panainte. The molen polymorphic processor. *Computers, IEEE Transactions on*, 53(11):1363–1375, Nov. 2004.

- [WG02] P. Wielage and K. Goossens. Networks on silicon: blessing or nightmare? *Digital System Design, 2002. Proceedings. Euromicro Symposium on*, pages 196–200, 2002.
- [WHS07] Pascal T. Wolkotte, Philip K. F. Holzenspies, and Gerard J. M. Smit. Fast, accurate and detailed NoC simulations. In *Network on Chips*, May 2007.
- [Wil08] A. Willig. Recent and emerging topics in wireless industrial communications: A selection. *Industrial Informatics, IEEE Transactions on*, 4(2):102–124, May 2008.
- [WN04] Andreas Weisensee and Darran Nathan. A self-reconfigurable computing platform hardware architecture. *CoRR*, cs.AR/0411075, 2004.
- [WNP04] Herbert Walder, Samuel Nobs, and Marco Platzner. Xf-board: A prototyping platform for reconfigurable hardware operating systems. In Toomas P. Plaks, editor, *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'04, June 21-24, 2004, Las Vegas, Nevada, USA*, page 306. CSREA Press, 2004.
- [WP04] Herbert Walder and Marco Platzner. A runtime environment for reconfigurable hardware operating systems. In Marco Platzner Jürgen Becker and Serge Vernalde, editors, *Field Programmable Logic and Application, 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004, Proceedings*, volume 3203 of *Lecture Notes in Computer Science*, pages 831–835. Springer, 2004.
- [Xil] Xilinx. <http://www.xilinx.com/products/devices.htm>.
- [Xil04] Xilinx. *Virtex II Platform User Guide*, 2004.
- [Xil05a] Xilinx. *Virtex II Platform User Guide*, 2005.
- [Xil05b] Xilinx. *Xilinx Development Systems Reference Guide*, 2005.
- [Xil06a] Xilinx. Early access partial reconfiguration user guide. Technical report, Xilinx inc., 2006.
- [Xil06b] Xilinx. *Partial Reconfiguration Software User's Guide*. Xilinx, 2006.
- [Xil07] Xilinx. *Virtex-II Platform FPGAs: Complete Data Sheet*, ds031 edition, 2007.
- [Xil08a] Xilinx. *Virtex-4 FPGA User Guide*, ds031 edition, 2008.
- [Xil08b] Xilinx. *Virtex-5 Family Overview*, ds100 edition, Sep. 2008.
- [YKR06] Eduardo de la Torre Yana Krasteva and Teresa Riesgo. Partial reconfiguration for core reallocation and flexible communications. In Gilles Sassatelli, Leandro Soares Indrusiak, Manfred Glesner, and Lionel Torres, editors, *Proceedings of the 2nd International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2006, Montpellier, France, July 2006*, pages 91–97. Univ. Montpellier II, 2006.
- [ZGZ07] L.Q. Zhuang, K.M. Goh, and J.B. Zhang. The wireless sensor networks for factory automation: Issues and challenges. *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 141–148, Sept. 2007.

Thesis Publications

6.7 Book Chapters and Journals

- Y. E. Krasteva, C. Papagianni, E. Kosmatos, E. de la Torre, I.S. Venieris and T. Riesgo, "ENAMORADO: Enabling Nomadic Agents in a Multimedia ORiented Architecture of Distributed Objects", in "Innovation and the Knowledge Economy", Edited by Paul Cunningham y Miriam Cunningham October, IOS Press 2005, p-p: 975-982, ISBN: 1-58603-563-0 ISSN: 1574-1230.
- Yana E. Krasteva, E. de la Torre, T. Riesgo, "Dynamic Reconfigurable NoC (DRNoC) Architecture for Commercial FPGAs. Application and Evaluation Through Emulation, Dynamic Reconfigurable Network-on-Chip Design: Innovations for Computational Processing and Communication, publisher: Information Science Reference Book, **abstract accepted, chapter under second review**.
- Yana E. Krasteva, Jorge Portilla, J.M. Carnicer, E. de la Torre, T. Riesgo, "Run-time Reconfigurable Nodes for Wireless Sensor Networks Applications", IEEE Transactions on Industrial Electronics, **under second review**.

6.8 International Conferences

- Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo, "Using partial reconfiguration for SoC design and implementation", to appear in Proceedings of the SPIE Symposium on Microtechnologies for the New Millennium (SPIE'09), Dresden (Germany), May 2009.
- Yana E. Krasteva, Jorge Portilla, José María Carnicer, E. de la Torre and T. Riesgo, "Remote HW-SW Reconfigurable Wireless Sensor Nodes" in Proceedings of the IEEE Annual Conference of the IEEE Industrial Electronics Society (IECON'08), pp. 2483-2488, Orlando-Florida (USA), November 2008.
- Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo, "Virtual Architectures for Partial Runtime Reconfigurable Systems. Application to Network on Chip based SoC Emulation", in Proceedings of the IEEE Annual Conference of the IEEE

Industrial Electronics Society (IECON'08), pp. 211-216, Orlando-Florida (USA), November 2008.

- Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo, "DRNoC, an On Chip Communication Solution for partial Runtime Reconfigurable Systems" in Proceedings of the International Workshop on Reconfigurable Communication-Centric System-on-Chips (ReCoSoC'08), pp. 207-210, Barcelona (Spain), July 2008.
- Yana E. Krasteva, Francisco Criado, Eduardo de la Torre and Teresa Riesgo, "A Fast Emulation-Based NoC Prototyping Framework", in Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig'08), IEEE 2008, pp. 211-216, Cancun (Mexico), December 2008.
- Yana E. Krasteva, Francisco Criado, Eduardo de la Torre, T. Riesgo, "NoC Emulation based on Partial Reconfiguration", in Proceedings of the International Conference on Design of Circuits and Integrated Systems (DCIS'08), Grenoble (France), November 2008.
- Jorge Portilla, Yana E. Krasteva, José María Carnicer and T. Riesgo, "Wireless Sensor Networks Node with Remote HW/SW Reconfiguration Capabilities", in Proceedings of the International Conference on Design of Circuits and Integrated Systems (DCIS'08), Grenoble (France), November 2008.
- Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo, "Reconfigurable Heterogeneous Communications and Core Reallocation for Dynamic HW Task Management" in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'07), pp.873-876, New Orleans (USA), May 2007.
- Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo, "Creating Partially Reconfigurable Systems", in Proceedings of the International Conference on Design of Circuits and Integrated Systems (DCIS'07), Sevilla (Spain), November 2007.
- Yana E. Krasteva, Didier Joly, Eduardo de la Torre and Teresa Riesgo "Virtex II Bitstream Manipulation: Application to Reconfiguration Control Systems", in Proceedings of the 16th IEEE Intl. Conference on Field Programmable Logic and Applications (FPL'06), pp. 1-4, Madrid(Spain), August 2006.
- Yana E. Krasteva, Eduardo de la Torre, Teresa Riesgo "Partial Reconfiguration for Core Reallocation and Flexible Communications", in Proceedings of the International Workshop on Reconfigurable Communication-Centric System-on-Chips (ReCoSoC'06), pp. 91-97, Montpellier (France), July 2006.
- Yana E. Krasteva, Eduardo De la Torre and Teresa Riesgo, "Applying Partial Reconfiguration for Debugging and Monitoring FPGA based Reconfigurable Systems", in Proceedings of the International Conference on Design of Circuits and Integrated Systems (DCIS'06), Barcelona (Spain), November 2006.
- Yana E. Krasteva, Ana B. Jimeno, Eduardo de la Torre and Teresa Riesgo, "Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs", in Proceedings of the IEEE International Workshop on Rapid System Prototyping (RSP'05), pp. 77-83, Montreal (Canada), June 2005.

- Yana E. Krasteva, Ana B. Jimeno, Eduardo de la Torre and Teresa Riesgo, "Flexible Core reallocation for Virtex II structures", in Proceedings of the International Embedded Reconfigurable Systems and Architectures (ERSA'05), pp.189-195, ACM 2005, Las Vegas (USA), June 2005.

6.9 National Conferences with Review Process

- Y. E. Krasteva, A. B. Jimeno, E. de la Torre and T. Riesgo, "Reubicación de bloques dentro de una estructura de slots para Virtex II: Aplicación a dispositivos embebidos adaptables", in Proceedings of Jornadas de Computación Reconfigurable y Aplicaciones (JCRA 05), Granada (Spain) 2005.
- X. Pena, Y. Krasteva, A. B. Jimeno, E. de la Torre and T. Riesgo, "ENAMORADO: Una experiencia hacia los terminales móviles reconfigurables en entornos multimedia" in Proceedings of Ubiquitous Computing and Ambient Intelligence (UCAI'05), Granada (Spain), September 2005.
- A. B. Jimeno, Y. E. Krasteva, E. de la Torre and T. Riesgo, "ENAMORADO: Reconfiguración Parcial de FPGAs en Dispositivos Móviles", in Proceedings of Jornadas de Computación Reconfigurable y Aplicaciones (JCRA 03), Madrid (Spain) 2003.

List of Symbols and Abbreviations

Abbreviation	Description	Reference page
API	Application Program Interface	112
ASICs	Application Specific Integrated Circuits	3
BMs	Bus Macros	29
BM-Xilinx	Xilinx defined Bus Macros	59
BS	Bitstream	118
BT	Block Type	121
BRAM	Embedded Block RAM memory	45
CLBs	Configurable Logic Blocks	10
CPLD	Complex Programmable Logic Device	38
CRC	Cyclic redundancy Checksum	120
CTG	Communication task graph	85
DRNoC	Dynamic Reconfigurable NoC	63
DHPs	Dynamic Hardware Plugins	31
EAPR	Early Access Partial Reconfiguration Flow	113
FDIR	Frame Data Input Register	120
FPGAs	Field Programmable Gate Arrays	5
FPOA	Field Programmable Object Array	14
FPX	Field Programmable Port eXtender	31
HDL	Hardware Description Language	29
HWOS	Hardware Operating Systems	33
HW	Hardware	21
IOBs	Input Output Blocks	10
IOs	Input Outputs	4
IPcores	Intellectual Property Cores	4
ISE	Integrated System Environment Software	113
LC	Logic Cell = 1 LUT+ 1FF + Carry Logic	49
LSB	Least Significant Bit	123
LUTs	Look Up Tables	10
NCD	Native Circuit Description	111
NI	Network Interface	89
MJA	MaJor Address	121
MNA	MiNor Address	121
MSB	Most Significant Bit	123
MUL	Embedded Multiplier	28

Abbreviation	Description	Reference page
NoCs	Networks on Chip	65
PARBIT	PARTial Bitfile Transformer	115
PAR	Placed And Route	111
pBS	partial BitStream	30
PE	Programming Elements	71
PPCs	Embedded microprocessor Power PC	124
pRTRS	partial Run-time Reconfigurable Systems	25
P2P	point to point	51
P2M	point to multi-point	51
RNI	Reconfigurable Network Interface	78
RN	Reconfigurable Node	79
RRM	Reconfigurable Routing Module	78
SM	Switch Matrix	11
SW	Software	33
TBUFs	Tri-State Buffers	10
TG	Traffic Generator	88
TR	Traffic Receiver	88
UCF	User Constraint File	56
VA	Virtual Architectures	25

La ciencia se compone de errores,
que a su vez, son los pasos hacia la verdad.

Julio Verne